RICE UNIVERSITY

Functional representation and manipulation of shapes with applications in surface and solid modeling

by

Powei Feng

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

Approved, Thesis Committee:

Joe Warren, Professor, Chair
Computer Science

Ron Goldman, Professor
Computer Science

Marcia O'Malley, Associate Professor
Mechanical Engineering

Houston, Texas

Apirl, 2013

Functional representation and manipulation of shapes with applications in surface and solid modeling

Powei Feng

Abstract

Real-valued functions have wide applications in various areas within computer graphics. In this work, we examine three representation of shapes using functions. In particular, we study the classical B-spline representation of piece-wise polynomials in the univariate domain. We provide a generalization of B-spline to the bivariate domain using intuition gained from the univariate construction. We also study the popular scheme of representing 3D density distribution using a uniform, rectilinear grid, where we provide a novel contouring scheme that culls occluded inner geometries. Lastly, we examine a ray-based representation for 3D indicator functions called ray-rep, for which we present a novel meshing scheme with multi-material extensions.

This thesis is by no means a solitary feat. My contribution to the work described in this thesis would not have been possible without the many people who have supported me through out my graduate career.

I thank my advisor Joe Warren for his constant guidance. Joe has taught me many things, and chief among them is to be persistent, tenacious, and meticulous in tackling problems. For any given problem, Joe challenged me not to blindly accept well-known ideas as the truth but to build my own understanding of the problem. For these and many other things I learned from him, I am truly grateful to Joe. I thank Ron Goldman. I learned the most from Ron while serving as his teaching assistant for various graphics courses. Ron's passion for teaching and his ability to effectively convey complex mathematical ideas encouraged my own exploration of crafting accessible presentations and easy-to-read writings. I thank Marcie O'Malley for her helpful advices in preparation for my thesis defense. Professor O'Malley's comment on the precision of speech is especially illuminating, and this advice will surely accompany me in my professional endeavours.

I also thank my collaborators in various projects. I thank Tao Ju for his tireless effort in the multi-material contours and contour culling projects. Tao taught me how to polish and focus research ideas and also how to effectively communicate these ideas in the written form. I thank Travis McPhail for his help and collaboration on our volume rendering project and for being a supportive officemate. I thank Cathy Lawson, Matt Baker, and Wah Chiu for giving me an opportunity to work on interesting biomedical projects and for providing me with various datasets. I thank Nira Dyn, Tim Warburton, and Scott Schaefer for their discussion on the biharmonic B-spline project. I also thank the anonymous reviewers for their help in polishing the three papers that are collected in this thesis.

I thank my father, mother, and brother for their great love and support through out my graduate years. Their patience and encouragements are invaluable in stretches

both tough and not. I am especially thankful that my family has been supportive and respectful of my decisions and never placed any unreasonable pressure on me. I thank my brothers and sisters at Fort Bend Community Church and in Rice Chinese Christian Fellowship. Without their prayers and support, I would not have had the moral, emotional, and spiritual strengths to brave through graduate school.

Lastly, I thank my lord and savior, Jesus Christ, who has shown me the wonders and beauty of his creation through computer graphics and beyond. I thank Jesus for walking with me in the times difficult and not. I pray that however little I may have contributed to human knowledge in this thesis, I have done it for His glory alone.

Contents

Illustrations

Chapter 1

Introduction

One of the primary concerns of computer graphics is to draw and manipulate two-dimensional (2D) or three-dimensional (3D) shapes of varying complexity. The first step in shape drawing is to model the shape using a description that is suitable for computation. For example, a 2D circle can be described with a point that indicates the center of the circle (two real-valued scalars) and the radius of the circle (one real-valued scalar). This description of a circle using three scalars is suitable for computation because the amount of storage is finite, and the representation of real numbers is a well-addressed problem in modern computing. In contrast, consider the description of a circle as an infinite set of points that is a fixed distance away from the center of the circle. This infinite description is not suitable for computation as modern computers cannot store an infinite amount of data. In computer graphics, representing shapes using computationally suitable description is commonly referred to as *shape* or *geometric modeling*.

In this thesis, our focus in shape modeling is on representing shapes using functions. The field of functional representation encompasses a wide range of applications. For example, techniques such as *B-splines*, or piece-wise smooth functions, have been used to build interactive modeling tools. B-splines can be easily implemented and efficiently evaluated, and they can be used to construct a wide range of smooth curves or surfaces. These and other properties make B-splines an attractive tool in shape modeling; popular programs such as AutoCAD® and Maya® have built B-spline capabilities into their toolsets. Furthermore, the B-spline basis functions and other radially symmetric functions (or radial basis functions, RBFs) are often used to

interpolate or approximate discrete data points in high dimensions. In graphics, data approximation using RBFs has been used to reconstruct polygonal structures from sets of scattered points [11]. These reconstruction techniques allow artists and movie makers to scan real-life objects and easily reproduce the objects in a digital context. Other graphics uses for B-splines and RBFs include interpolating or approximating a series of shapes for animation. A common technique in animation is called key-framing, where the artist provides a small number of fixed poses for certain discrete points in time, and interpolation is performed for the in-between times. B-splines and many other RBFs are well-suited for this task due to their smoothness [69]. Beyond graphics, B-splines and RBF techniques have been widely used in other fields of industry and academia [9]. In this thesis, we present a recent development that integrates techniques of RBF approximation with the construction of the B-spline basis functions. Our work on B-spline construction can be applied in the familiar fields of data approximation in Euclidean domains but also in domains that are more general than previously possible.

Another use of functional description is found in physical imaging. Biological and medical research have led to multiple imaging technologies that can capture information beyond what is visible with the human eye. One such example is the X-ray Computed Tomography (CT) [37]. CT can capture internal structures of an object that are not typically visible; an example would be the bones within a human body. Other similar technologies include Magnetic Resonance Imaging (MRI) and Electron Microscopy (EM) [74, 8]. These imaging technologies produce information in three dimensions, and typically, the information is converted into scalar values associated with a 3D, rectilinear, uniform grid. Intuitively, these scalars can be viewed as discrete samples of a function in 3D. This type of data is commonly known as volume data or *density maps*; we use the latter term for the presentation in this thesis. The most common graphical task in working with a density map is to visualize the map, and techniques have been developed to provide various flavors of visualization. Pop-

ular technique such as direct volume rendering (DVR) and its many variants allow the user to assign opacity to values of the 3D function [18]. Other techniques such as maximum intensity projection (MIP) renders the maximum value of the 3D function along the viewing direction [58]. One popular form of visualization is to render level-sets of the 3D function [54]. If we consider the MRI of a human hand as the density map, then one level-set of this map corresponds to skin and flesh of the hand while another level-set corresponds to the bones. One common problem in rendering level-sets within a density map is that a level-set might contain surfaces that are not visible when viewed far away from the object. These invisible, extracted surfaces can slow down the rendering speed. In this thesis, we propose a method that *culls* out the invisible surfaces from the level-sets of a 3D density map, which allows for faster rendering speed.

Aside from computer graphics, functional descriptions for shapes have also found many uses in mechanical engineering. Modern machining and manufacturing processes take a computational description for an object and transform it (most commonly by programming a machine) into a physical object. One example of this transformation from description to object is numerical control milling (or NC milling). A milling machine consists of several apparatuses that rotate and position a solid while a drill cuts away at the solid to form a shape. An NC milling machine is simply a computer controlled milling machine. Ray-rep (or dexels) is a description is that is born out of the NC milling process [55, 39, 56]. A ray-rep is a set of parallel rays, where each ray is associated with points that denote the transition from empty space to non-empty space and vice-versa. Ray-rep is well-suited for performing set operations (such as union, intersection, or complement) on shapes. (Using set operations on shapes to create new shapes is referred to as constructive solid geometry or CSG). CSG with piece-wise linear primitives (such as a collection of triangles) or with more general piece-wise functions (such as splines) remains a very difficult problem with no known direct and efficient solutions. CSG with density maps and ray-reps, however,

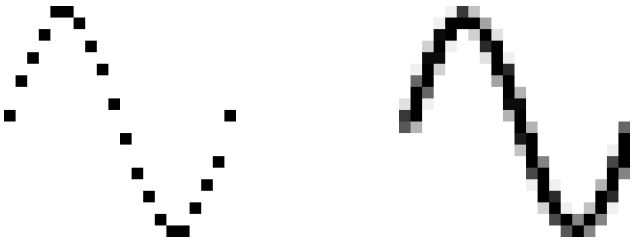
is a simple task of performing of set operations on the individual voxels or rays [70]. Additionally, ray-reps typically require less memory or storage than density maps. These two properties make ray-rep an attractive alternative in certain graphics or engineering applications. In our work, we consider the problem of visualizing ray-reps by surface extraction. Much like using marching cubes for density maps, our approach is intuitive and simple to implement. Moreover, we show how extracting surfaces from a multi-labelled ray-rep (beyond empty and non-empty spaces) can be extended from our two-material extraction scheme.

Brief examples

In the following, we show a few examples of how functional representations are used in graphics. In the simplest terms, a function is a mapping that takes elements in one space (the domain) to elements in another space (the codomain). To illustrate how functions can be used in graphics, we consider a simple example of drawing a curve on the 2D computer display. Intuitively, a computer display is a collection of 2D regularly spaced, colored dots called *pixels*. To draw a curve on the screen, we describe a function that maps elements from the horizontal coordinate space of the pixels onto the elements of the vertical coordinate space of the pixels. Assume the screen is of size $S_x \times S_y$. The horizontal pixel coordinates is represented as a bounded subspace of the integers (i.e. the range $[1, S_x]$), and the vertical pixel coordinates is similarly defined as $[1, S_y]$. We can then draw a set of pixels that approximates a smooth curve by writing a function $f_{\text{ex}}(x) = \sin\left(2\pi \frac{x-1}{S_x-1}\right)$ and computing the set of pixels

$$\{(x, f_{\text{ex}}(x)) | x \in [1, S_x]\} \cap ([1, S_x] \times [1, S_y]).$$

Figure 1.1a shows this curve for a screen size of 50×50. This example is just one of many ways of using a functional definition to model shapes in graphics. In this thesis, we examine three different representations of functions in graphics, and we



(a) (b)

Figure 1.1 : A simple function ($\sin(x)$) rendered on a 50×50 display (a). A smoothly rasterized version of the function ($\sin(x)$) (b). Antialising is applied in (b).

present novel algorithms that enable more flexibility for each representation in practical graphics applications.

In our example above, we have assumed that the domain and codomain spaces are subsets of the 1D integers. This definition is limiting in two aspects. First, the 1D integers is a discrete domain, and we have chosen such a discrete domain because the modern computer is only capable of computing with discrete values. However, if we separate the notion of discretization (using only integers) from the notion of representation, we can represent shapes using a much richer set of functions over the real numbers. Second, since the domain and the codomain are both 1D, the set of shapes that we can represent using one-to-one functions is very limited. For example, we cannot represent a closed curve in 2D under this definition.

The first limitation is typically addressed by rasterization, where shapes are represented in a continuous domain and are then discretized and made visually pleasing using discretization methods. Figure 1.1b shows a visually more appealing rasterization of the sine curve. Our work does not cover the well-studied problem of rasterization; we encourage the interested reader to seek out standard graphics texts for more information [27]. For the rest of our presentation, we assume that rasterization methods are readily available, and we primarily focus on shapes in continuous domains.

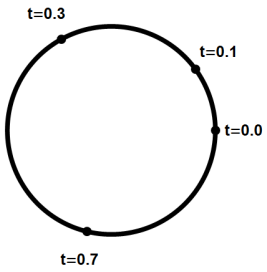
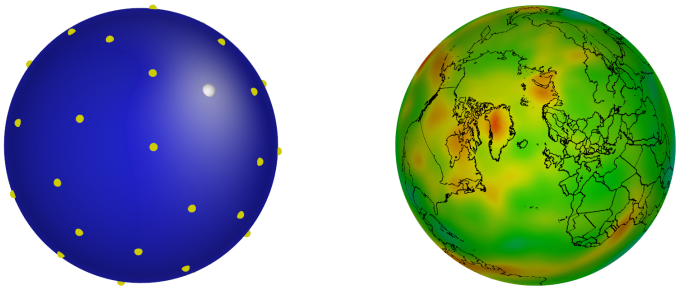


Figure 1.2 : A parametric representation of the 2D circle. Different parameter values (t's) correspond to different points on the circle.

For the second limitation, we look more closely at parametric representations that allow us to model more interesting shapes. A parametric curve in 2D has the form,

$$f(t) = (x(t), y(t))$$

where $x(t)$ and $y(t)$ are functions that maps from the 1D real numbers to the 1D real numbers. As an example, a circle can be represented as $f(t) = (\cos(t), \sin(t))$. More generally, closed curves can be represented by defining the parametric function over a cyclic domain. Figure 1.2 shows this parametric curve, and points on the curve are associated with different parameter values. Our task in geometric modeling is to identify good candidates for functions $x(t)$ and $y(t)$. In practice, we are mainly interested in tasks that involve *interactive* creation and manipulation of *smooth* curves and surfaces, such as designing a automobile or drawing a cartoon. A good creation and manipulation scheme for shapes should allow for *local control* of the shape since a user's attention is most often focused on modifying the local rather than the global shape. The requirement of local control, smoothness, and interactive manipulation reduces the space of suitable parametric functions. One of the most popular solutions is to take a small set of points in 2D and design a parametric curve that follows closely to the points. This small set of points is often called control points. Since the number of control points is small, the user is allowed to interactively manipulate the positions



(a) (b)

Figure 1.3 : Chapter 3 covers the derivation of bivariate B-splines (biharmonic B-splines) basis functions on the sphere (a). Biharmonic B-splines can be used to define smooth functions on the sphere; modelling carbon distribution over the earth is one possible application (b).

of control points, which addresses the interactive manipulation requirement. For smoothness and local control, we consider the following formulation of control point curve design,

$$f(t) = \sum_{i=1}^{n} p_i B_i(t),$$

where p_i's are n control points (for our example, points in 2D) and $B_i(t)$ are functions associated with each control point.

The most popular choice of functions (the B_i's) in industrial design is the B-splines basis functions. B-splines are piece-wise polynomials that are locally supported with guaranteed continuity at joints where the pieces meet (these properties are more carefully defined in later chapters). Thus, the smoothness and local control requirements are both met by B-splines. Applied research of B-splines stretches back five to second decades, and B-splines have been proven to be a powerful tool in shape design and in data approximation. Most work on B-splines, however, focus on the univariate construction, where the domain is the 1D real numbers. A few attempts have been made in generalizing B-splines to 2D, but these previous generalizations do not retain

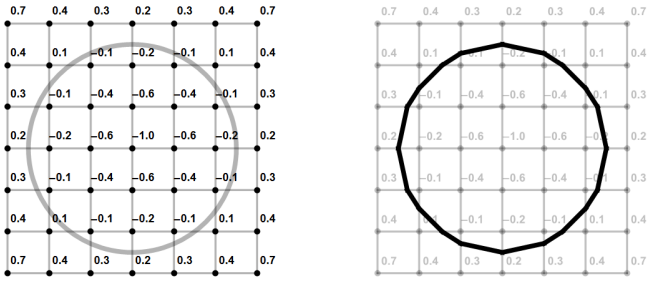
all of the attractive properties of the 1D B-splines. In this thesis, we present a derivation for B-splines that can be extended to 2D; this derivation retains many of the benefits of the 1D B-splines. In Chapter 2, we describe the standard formulation for 1D B-splines (where the domain is the 1D real numbers) and provide an alternative derivation that involves solutions to a differential equation. This differential formulation enables us to define B-splines over the bounded 1D domain with the ability to reproduce linear functions. In Chapter 3, we extend the differential construction to two-dimensions. This extension leads us to a novel class of bivariate B-splines, which we call the *biharmonic B-splines*. We show that this definition allows us to build B-splines over 2D-manifolds, which we demonstrate by building B-splines over the sphere (see Figure 1.3).

B-splines and other parametric constructions are useful for building and manipulating shapes such as curves. Parametric representations are general enough such that they can represent both closed and open curves. In terms of closed curves, however, the parametric definition cannot readily provide us with information about the partition of space by a closed curve, or more intuitively, which points in the codomain are inside the curve, and which points are outside of the curve. Representations that are able to describe this notion of partition is then describing a solid or a volume (as oppose to a curve or a surface). Quite a few representations are available for modeling solids, and the last two chapters of this thesis describe two such representations.

One representation for a 2D solid is to describe the solid region using a function that maps from the 2D real numbers to the 1D real numbers,

$$g(x, y) = c,$$

where c is a real number that corresponds to the space partitioning curve. Without loss of generality, we can define the points that are interior to the curve as $g(x, y) < c$, and points that are exterior to the curve as $g(x, y) > c$. For example, a solid bounded by the unit circle centered at the origin is defined by writing $g(x, y) = x^2 + y^2$ and $c =$



(a) (b)

Figure 1.4 : Density map of the 2D circle (a), and the 0-level contour extracted from the density map (b).

1. In this fashion, we can design a host of shapes just by writing different boundary functions $g(x, y)$. However, trying to model complex shapes using analytic functions (such as polynomials) as $g(x, y)$ can be too restrictive and expensive. Instead, most solid modeling applications uses the "piece-wise" concept as in splines.

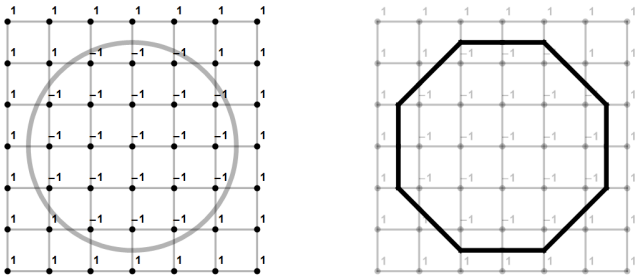
A piece-wise solid representation defines the function over a uniformly spaced rectilinear grid. Each grid point corresponds to a sample of the function, and function values for non-grid points are interpolated from grid point function values. These piece-wise solid functions are also referred to as density maps. Figure 1.4a shows the density map for the 2D circle function $g(x, y)$. The name of density map comes from the fact that sensing technologies, such as Magnetic Resonance Imaging, produce functions that measure the density of the imaged object. Density maps are amenable to shape manipulations. In particular, constructive solid geometry (CSG) with density maps can be easily accomplished by taking minimum and maximums of functions. Additionally, many techniques are available for fast rendering of density maps. Two of the well-known methods are the Marching Cubes (MC) and direct volume rendering (DVR) [54, 18]. MC examines a neighborhood of eight adjacent density values (called a cube) in the density map grid and extracts the boundary surface in that eight-value

(a) (b) (c)

Figure 1.5 : A level-set of a tooth dataset extracted using Marching Cubes from a density map (a). Transparent rendering of the level-set shows that the level-set contains inner geometries that are not visible when viewing from the outside (b). Our method reduces (or culls) the invisible geometries to enable faster rendering (c).

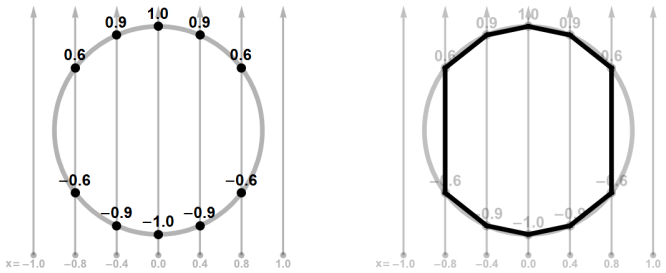
neighborhood. The MC algorithm repeats this process for all of the cubes in the density map. The result is a collection of piece-wise linear surface patches that interpolates the given density map. MC has been embraced by academia and industry due to its simplicity and speed. Figure 1.4b shows an example of the 2D marching cubes on the circle density map. However, classical marching cubes does not consider the task of culling invisible surfaces in its extraction. In Chapter 4, we discuss this problem of extracting only visible surfaces from density maps, and we provide a novel method that culls out non-visible surfaces for faster rendering of density maps. Figure 1.5 briefly illustrates the result of our method on a tooth dataset.

The density map representation of a solid contains more information than just a partition of space; this description can be used to describe distance to a boundary curve or, as mentioned, the densities of regions in the codomain. However, in many applications, our primary interest is the partition of space into solid and non-solid. We can interpret this type of division as a specialized density function by considering a function that yields -1 in the solid region and 1 in the non-solid region. This



(a) (b)

Figure 1.6 : An indicator function of the 2D circle (a), and 0-level contour extracted from the indicator function (b).



(a) (b)

Figure 1.7 : Ray-rep of the 2D circle (a), and a contour extracted from the ray-rep (b).

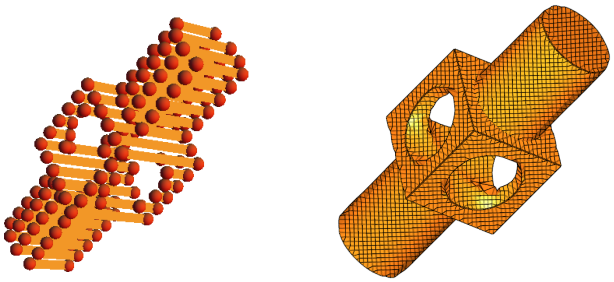
function is inherently non-analytic and is sometimes referred to as the indicator (or characteristic) function. Under this representation, a solid bounded by a unit circle centered at the origin is written as,

$$h(x,y) = \left\{ \begin{array}{ll} 0 & \text{if } x^2+y^2 = 1 \\ 1 & \text{if } x^2+y^2 > 1 \\ -1 & \text{if } x^2+y^2 < 1 \end{array} \right\}.$$

Once again, to represent complex shapes, we can adopt a piece-wise approach by taking samples of the indicator function at regular grid points and performing inter-

polation for points in between the grid points. Figure 1.6a shows an example of the piece-wise indicator function for a circle, and Figure 1.6b is a contour extracted from the piece-wise approximation that uses a bilinear interpolation scheme (2D MC). Note that, in general, the piece-wise indicator function contains redundant information. A more compact representation is to record the beginning and ending vertical coordinates of the solid for each vertical grid line. Moreover, for the circle, we can store the intersections of the true circle (as oppose to a piece-wise linear approximation) with the grid lines for even higher accuracy in representing the circle. Figure 1.7a shows an alternative representation; note that it is a collection of vertical rays, and values along each ray mark the intersections between circle and the ray. This representation of storing intersection points along parallel rays is called ray-rep (short for ray-representation). Figure 1.7b shows the extracted contour of the ray-rep for the 2D circle. Ray-rep has a considerable history in mechanical engineering, and its counterpart in computer graphics (known as layered depth images) has also been used in various applications [55, 39, 56, 72]. Most contour extraction algorithm for ray-rep requires converting ray-rep to density maps and performing marching cubes for extraction. The extra layer of conversion introduces inaccuracies in the final result. For the last chapter of this thesis, we review a direct approach to extract polygonal contours from ray-rep. We demonstrate that having hermite data (normals) at the intersection points can generate higher quality surfaces. Furthermore, we extend the algorithm for two-material (binary partitioning) ray-rep contouring into the multi-material domain (n-ary partitioning). Figure 1.8 shows an example of an ray-rep of the popular "mechpart" model, and a mesh reconstruction of the model is shown on the right-hand side.

In this thesis, we study three well-known representation of shapes using functions in computer graphics: B-splines (Chapters 2 and 3), density maps (Chapter 4), and ray-reps (Chapter 5). For each representation, we present novel contributions in the construction, manipulation, and visualization of these representations. For each con-



(a) (b)

Figure 1.8 : A ray-rep of the "mechpart" 3D model (a). A surface mesh extracted from a ray-rep using our 3D reconstruction method (b). Please be aware that the ray-rep in (a) is simplified for illustration purposes; the actual ray-rep that generated (b) is a denser sampling of the model.

tribution, we demonstrate how our improvements over previous work can be applied in practice.

The work presented in this thesis are gathered mostly from three published articles. The biharmonic B-splines work was published in the conference proceedings of SIGGRAPH 2012 [23]. The contour culling work was published in the conference proceedings of Shape Modeling International (SMI) 2011 [22]. The ray-rep meshing work was published in the conference proceeding of International Symposium on Visual Computing (ISVC) 2012 [24].

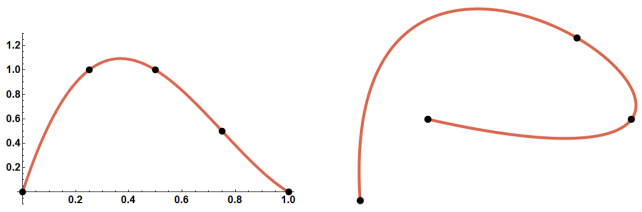
Chapter 2

Univariate B-splines

2.1 Motivation

Advances in the speed of computing have enabled computer softwares to displace much of the traditional pen-and-paper based design paradigm. For our focus of graphics and shape modeling, computer-aided methods has many strengths over the manual methods, which include: numerical precision, efficiency, and compatibility with numerical analysis. Due to these strengths, computer-aided shape modeling methods have flourished, and computer programs such as 3D Studio Max®, Maya®, AutoCAD®, and Photoshop® are now the industry standards for shape design. In this chapter we look at the specific problem of curve modeling and study how polynomial approximation methods have been used to model curves.

In computer graphics, the embedding space for a shape is typically the two-dimensional



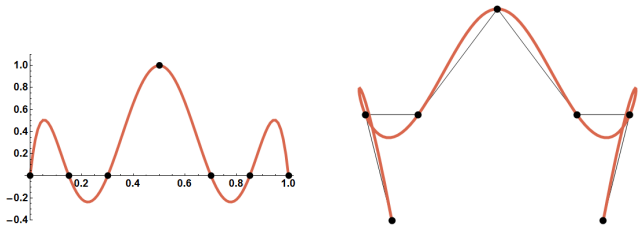
(a) (b)

Figure 2.1 : Smooth interpolation with polynomials. Functional univariate polynomial interpolation with the Lagrange method (a). 2D curve constructed with Lagrange interpolation (b).

(2D) or three-dimensional (3D) real numbers. For simplicity, we loosely define a shape as a connected and closed subset of the 2D or 3D reals. Under this definition, a curve, a sphere, or a car would all be examples of shapes. In this chapter, we cover one of the main components in curve and surface design called splines, or piece-wise polynomial functions. We review the construction of one particular univariate spline, called the B-spline, which has been widely embraced in industry and academia [21]. This review of univariate B-spline serves as the foundation to the bivariate extension of the B-spline, which is covered in the next chapter.

The examples of this chapter focus on the modelling of 2D curves with B-spline. (However, we note that B-spline can be easily used to generate a 2D surface by ways tensor products [21]). 2D curves can be used in applications such as designing fonts, creating cartoons, or providing simple pictorials for webpages or presentations. From these common applications, we infer two general principals for curve design: the curve should be smooth for reasons of aesthetics and manipulating a curve should be an interactive and easy process. One design method that satisfies these guidelines is to provide a set of points (control points) that describes the general shape of the curve and algorithmically compute a curve that runs through (or interpolates) those points (see Figure 2.1). Under this method, we can achieve interactive and efficient modeling by limiting the number of control points to a humanly manageable size. Furthermore, we can satisfy the aesthetics principal by requiring the interpolating curve to meet certain smoothness constraints. We refer to this method as *smooth interpolation*.

Before we delve into the details of interpolation, note that we can reduce the problem curve modelling in 2D into 1D by solving two 1D problems with respect to each of the coordinate spaces. Given this context, curve modelling is equivalent to performing smooth interpolation in 1D, which can be described as thus: we are given a set of m pairs of values in the real-valued domain and codomain, $(t_0, y_0), (t_1, y_1), \ldots, (t_{m-1}, y_{m-1})$,



(a) (b)

Figure 2.2 : Oscillation behavior of high-order polynomial interpolation. A 2D example of lagrange interpolation with high oscillation behavior.

and we want to find a *smooth* function $f(t)$ such that

$$f(t_i) = y_i. \tag{2.1}$$

By smoothness, we mean that $f(t)$ is continuous and that lower order derivatives of $f(t)$ are continuous. Many work on functional interpolation have focused on polynomial interpolating functions. Polynomials are simple to analyze in terms of its differentiability. Additionally, computer evaluation methods for polynomials have been extensively studied, and various fast evaluation and approximation methods have been proposed for low-order polynomials [33]. However, when we are given a large number of data points, the associated interpolating polynomial tend to have high degrees. Evaluating high degree polynomials on the computer can result in inefficient and inaccurate results. Furthermore, polynomial interpolation is known to have an oscillating behavior for large number of interpolation points (see Figure 2.2).

Splines is a class of interpolating functions that are able to address the shortcomings of polynomial interpolation. Splines consists of joining piece-wise polynomial functions to form a continuous function. Typical spline techniques offer the user various degrees of smoothness at the joints of the splines. Splines have the strength that the degree of the polynomials used in the scheme is independent of the number of data points; splines are also simple to evaluate because each piece is a low degree polyno-

mial. Even with splines, the interpolation constraint in Eq. 2.1 is still too restrictive, and interpolating splines can still exhibit the oscillating behavior as seen in polynomial interpolation. In practice, users of splines often use *approximation* as oppose to interpolation to ameliorate the oscillating behavior of polynomial interpolation. Eq. 2.1 is then replaced with,

$$f(t_i) \approx y_i. \tag{2.2}$$

We can rephrase Eq. 2.2 in terms of curve modelling with a set of control points. When modelling a curve, a user typically wants the curve to follow the control points closely. We are given a sequence of m scalar values $t_0 \leq t_1, \leq, \ldots, t_{m-1}$, called *knots*. Knots correspond to m control points p_i's, and we require that $f(t_i) \approx p_i$. In other words, if we consider a parameter value (or a domain point) t_i and we look at the corresponding curve point $f(t_i)$ (a codomain point), we expect that the point on the curve to be very close to the control point p_i. Intuitively, if the control points are all relatively far apart, then we expect the curve point $f(t_i)$ to be closer to p_i than any other control point. We can further clarify our objective by writing the function f as the following,

$$f(t) = \sum_{i=0}^{m-1} p_i B_i(t), \tag{2.3}$$

where B_i are functions whose domain and codomain are both the 1D real numbers. We refer to the B_i's as basis functions. All polynomial interpolation or approximation schemes (piece-wise or not) can be written in the same form as Eq. 2.3, and this form gives us insight into the design of the basis functions B_i's. First, if we consider evaluating f at a knot value t_i, then we expect that $f(t_i)$ is close to the point p_i. This closeness property implies that the value of $B_i(t_i)$ should be a relatively large number less than or equal to 1. In addition, for all other basis functions B_j where $j \neq i$, $B_j(t_i)$ should be a relatively small value close to or equal to zero. We can deduce, then, that a bump-like (or bell-shaped) function that is centered over the knot t_i and decays to zero everywhere else is a good candidate for a basis function (see blue curve in

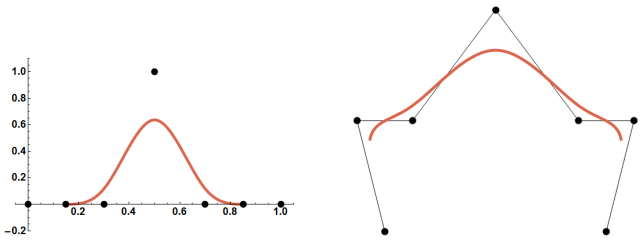
Figure 2.4). We refer to functions that satisfy this property as having *bump-likeness*.

Continuing with our curve modeling analogy, a user typically manipulates one control point at a time to incrementally adjust the shape of the curve. In most cases, the user's intent is to manipulate a local portion around the control point and not to change the entire curve. Hence, a good curve modeling framework should also provide *local* control. In terms of the basis functions, this local control implies that each basis function should evaluate to effectively zero some distance away from the center of the basis. We refer this criteria as *locality*.

One of our purposes in this chapter is to show that the B-spline is a good candidate set of functions that satisfy the our listed, desirable modelling properties. In practice, B-spline has been extensively studied and has been shown to have many other well-behaving properties that are suited in various applications. With respect to computer graphics, B-spline is frequently used for curve and surface modeling and approximation tasks such as time-varying pose approximation. Figure 2.3 shows examples of approximation with B-splines. Note that the B-splines approximation does not exhibit the oscillation problem seen with polynomial interpolation (Figure 2.2).

Each B-spline basis function B_i is a piece-wise polynomial where the each polynomial piece resides in the interval between the knots. For a degree d B-spline, each polynomial piece can be infinitely differentiated (or C^∞), and the each basis function is C^{d-1} at the values where the pieces meet, which are the knots. Our exposition does not cover the continuity analysis of the univariate B-spline; our focus is on the *bump-likeness* and *locality* of the B-spline. We encourage the reader to seek out classical texts by de Boor and Farin for more information on the continuity of the B-spline [15, 21].

In this chapter, we first explore the classical B-spline derivation, and we show that it has an equivalent view as solution to a partial differential equation. We show that the B-spline basis functions satisfy bump-likeness and locality. The second half



(a) (b)

Figure 2.3 : Approximation with cubic B-splines. Functional univariate B-spline approximation. 2D curve constructed with B-splines.

of this chapter examines the construction of non-uniform, one-dimensional B-spline over bounded domains. By studying conditions on the differential derivation, we are able to construct bounded B-splines that can reproduce linear functions. With this chapter as the foundation in the univariate case, we study the bivariate extension in the next chapter.

2.2 Univariate B-spline construction

Typical presentations of B-spline can be put into two categories: recurrence and divided difference. In the recurrence exposition, higher order B-spline basis is written as an affine combination of lower order basis. Looking at B-spline as a recurrence has given rise to many insightful analysis of B-splines. In particular, studies on blossoms, or polar forms, have provided a strong connection between evaluation techniques (de Boor's algorithm and knot insertion) and B-spline continuity analysis [68, 33]. Since our construction is more closely related to the divided difference construction, please refer to the references for more details on the recurrence construction of splines.

One of the earliest definitions of B-spline is by means of divided differences and truncated power functions [15]. For our exposition, we look at the definition of the cubic B-spline basis but note that the univariate construction described can be

extended to any degree. In addition to the construction, we show that B-spline indeed satisfies the properties of locality and bump-likeness. If the knots are uniformly spaced, then the basis function is centered over t_2. The blue curve in Figure 2.4 shows the shape of the cubic B-spline basis function.

We begin with a set of five knots $t_0 \leq, \ldots \leq t_4$. The following equation writes the cubic B-spline basis function over the knots $t_0, \ldots, t_4$ as the fourth difference of a third-degree, one-sided power function.

$$B(t) = (t_4 - t_0)(x - t)_+^3[t_0, \ldots, t_4] \tag{2.4}$$

The subscript $(x - t)_+^3$ indicates that the function is zero for $x \leq t$ and $(x - t)^3$ otherwise. The notation $f[t_0, \ldots, t_4]$ is the divided difference of the function f over the knots $t_0, \ldots, t_4$. The normalizing factor $(t_4 - t_0)$ ensures that the B-spline basis functions sum to the unity.

The locality property of B-spline can be shown by considering the cases where $t < t_0$ and $t > t_4$. If $t < t_0$, then $(x - t)^3 = (x - t)_+^3$ in the range $[t_0, t_4]$. The fourth divided difference of $(x - t)^3$ is 0. Hence, if $t \leq t_0$, then $B(t) = 0$ in the range $[t_0, t_4]$. If $t > t_4$, then $(x - t)_+^3 = 0$ in the range $[t_0, t_4]$, and the fourth divided difference of the constant function is also 0. Therefore, if $t > t_4$, then $B(t) = 0$. So we have concluded that $B(t)$ is localized to the range $[t_0, t_4]$.

We have shown that B-spline basis function are local, and we discuss the bump-likeness property of B-spline in the next section by studying an equivalent differential derivation of B-spline.

2.2.1 Differential view of B-splines in 1D

We proceed by studying a differential equation and note the equation's relation to the divided difference construction of B-spline. Next, we show that we can form the B-spline basis functions by taking linear combinations of the solutions to the differential

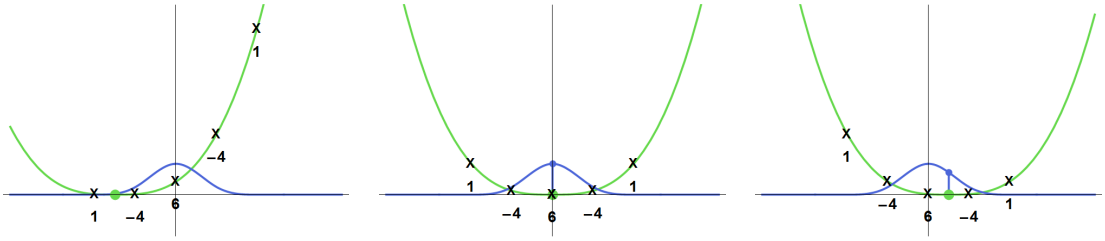


Figure 2.4 : Illustration of the differential construction of the univariate B-spline basis. The B-spline basis function (blue curve) is computed by taking the linear combinations of samples of the Green's function (green curve) at the knots and weighted by the difference of the third divided difference (values below the green curve).

equation. Under this construction, we expound upon the bump-likeness property of B-spline.

First, we consider the differential equation of the form

$$\Delta^2 \phi_t(x) = \delta(x - t), \tag{2.5}$$

where $\delta(x-t)$ is the Dirac delta centered at t, and Δ^2 is the fourth derivative. Equation of this form are called harmonic equations, and bi-, tri-, and similar prefixes indicates the order of the Laplacian operator (Δ). A solution to this biharmonic equation is called the fundamental solution, or in the presence of boundary conditions, the *Green's function*. We consistently refer to this solution (with or without boundary) as the Green's function. In the univariate case, the Green's function has the form,

$$\phi_t(x) = (1/12)|x - t|^3. \tag{2.6}$$

Note that the Green's function is the positive side of the power function defined in Eq 2.4 mirrored over the origin. Our next step is to connect the divided difference B-spline definition with the Green's function. Using the Green's function , we can write a new function

$$B_g(t) = (t_4 - t_0)\phi_t(x)[t_0, \ldots, t_4]. \tag{2.7}$$

We show that the B-spline basis function $B(t)$ is equivalent to $B_g(t)$ up to a constant multiple. We can rewrite $\phi_t(x) = (1/12)(2(x-t)_+^3 - (x-t)^3)$. Since $(x-t)^3$ is a third-degree polynomial, the fourth difference would annihilate $(x-t)^3$. Hence,

$$\begin{aligned} B_g(t) &= (t_4 - t_0)\phi_t(x)[t_0, \ldots, t_4] \\ &= \frac{t_4 - t_0}{6}(x-t)_+^3[t_0, \ldots, t_4] \\ &= \frac{1}{6}B(t) \end{aligned}$$

Thus, we have shown that $B_g(t)$ and $B(t)$ are equivalent definitions of B-spline. Furthermore, Eq. 2.7 indicates that the B-spline basis function can be written as the divided difference of the Green's function. Under this definition, we can alternatively write the B-spline as,

$$B(t) = \sum_{i=0\ldots4} n_i \phi_t(t_i). \tag{2.8}$$

In this form, the B-spline basis function centered over t_2 is written as a linear combination of samples of the Green's functions with some weights n_i. n_i is the mask associated with the difference of the third divided difference (or $(t_4 - t_0) \bullet [t_0 \ldots t_4]$). Figure 2.4 illustrates how the B-spline is formed by taking weighted samples of the Green's function over the knots.

We can now show bump-likeness by taking the finite integral of Eq. 2.5,

$$\mathbb{Z}_{t_0}^{t_4} \Delta^2 \phi_t(x) dx = \mathbb{Z}_{t_0}^{t_4} \delta(x-t) dx \tag{2.9}$$

Note that if $t < t_0$ or $t > t_4$, then the right-hand side of Eq. 2.9 is 0; otherwise, the right-hand side is 1. By setting the left-hand side of Eq 2.9 as a function dependent on t, that function becomes the characteristic function for the range $[t_0, t_4]$, which means that the function is 1 if $t \in [t_0, t_4]$ and 0 otherwise. We now consider the discrete form of Eq. 2.5. If we discretize the integral and fourth derivative of the left-hand side, then we would arrive at Eq. 2.8, where

$$\sum_{i=0\ldots4} n_i \phi_t(t_i) \approx \mathbb{Z}_{t_0}^{t_4} \Delta^2 \phi_t(x) dx$$

Therefore, we see that the B-spline basis function B is a discrete approximation to the characteristic function over the range $[t_0, t_4]$, which implies that B is bump-like over that same range.

2.3 Bounded B-spline

In the previous section we implicitly assumed that the knots are given over an infinite or circular domain. This section we examine construction for the bounded domain. Typically, bounded B-splines are constructed by duplicating knots at the two ends of the domain [15]. We take an alternative approach by framing the problem in the differential view. We describe a method for computing discrete B-spline basis for the bounded domain. We also briefly analyze the differences between knot multiplicity and the differential bounded splines.

First, we write the B-spline basis functions in matrix notation,

$$\mathbb{B} = N\Phi \tag{2.10}$$

where the columns of Φ are translates of the Green's function, and the rows of N are the divided difference masks associated with each basis. The rows of $\mathbb{B}$ are B-spline basis functions. In the unbounded case, N is an infinite matrix and $\mathbb{B}$ and Φ are infinite rows. In the bounded case, N is a finite, banded matrix. For the interior knots (knots that are at least two knots away from the boundary), their associated masks are simply the difference of the third divided difference over the knots. The choice for the difference masks along the boundary depends on boundary conditions of the differential equation. Furthermore, the Green's function associated with the bounded domain also depends on the boundary conditions, and hence, we can no longer use Eq. 2.6 as the general solution for Eq. 2.5.

Fortunately, we can use *refinement* (also known as knot insertion) to arrive at discrete B-spline basis functions. The idea of refinement is simple: given a spline defined by

a set of knots and a set of control points, we want to insert new knots into the spline and compute new control points such that the new spline is equivalent to the old spline. One interesting property of B-spline knot insertion is that the control points converges to the spline itself [15]. In other words, if we are only interested in a discrete B-spline curve (where a dense set of points approximates the actual piece-wise polynomial curve), then we can repeatedly insert knots using refinement to compute such a discrete spline.

For our bounded differential construction, we do not know the explicit formulas for the Green's functions associated with a bounded domain. Hence, we also do not have closed-form formulas for the B-spline basis functions. However, as the work in Chapter 3 shows, we do have a refinement scheme for this differential construction that involves solving a set of linear equations. We refer the reader to Chapter 3 for more details on the derivation of refinement; here, we proceed with the linear system setup without further qualification. Given a dense set of knots T_k and a coarse set of knots T_0, where $T_0 \subset T_k$. Let their associated difference masks be N_0 and N_k, and the associated coefficients (or control points) are p_0 and p_k. Let the associated B-spline basis functions be $\mathbb{B}_0$ and $\mathbb{B}_k$. The refinement equation states that if

$$N_k p_k = U_{k0} N_0 p_0, \tag{2.11}$$

$$a_k p_k = a_0 p_0. \tag{2.12}$$

then $\mathbb{B}_0(x)p_0 = \mathbb{B}_k(x)p_k$. U_{k0} is an upsampling matrix that inserts rows of zeros into the matrix N_0 for knots in T_0/T_k. The equation $a_k p_k = a_0 p_0$ ensures that the integrals of the fine and coarse B-spline functions are preserved. This linear condition eliminates the nullspace in the difference matrix and guarantees a unique solution to the linear system. The discrete B-spline basis $\mathbb{B}$ centered over knots T_0 and sampled over a dense knot set T_k can computed by solving the linear system

$$\begin{aligned} N_k \mathbb{B} &= U_{k0} N_0 \\ a_k \mathbb{B} &= a_0 \end{aligned} \tag{2.13}$$

Note that $\mathbb{B}$ is a matrix of dimension $|T_k| \times |T_0|$, where each column i of $\mathbb{B}$ is a discrete B-spline basis associated with knot $i \in T_0$; the spline is discretized over the fine knots T_k. Given this formulation, we observe that the discrete B-spline basis over a bounded domain requires the construction the bounded difference matrix for both the fine and coarse knot sets. To construct the difference matrices, we need to identifying the difference masks for the boundary knots that satisfy some boundary conditions. In the following section, we consider a derivation of the boundary masks values.

2.3.1 Linear reproduction constraints

Without loss of generality, we assume that the knots are uniformly spaced. For the interior knots, the difference mask is the sequence $(1, -4, 6, -4, 1)$. We write the difference matrix as

$$N = \begin{pmatrix} u_0 & u_3 & 1 & 0 & 0 & \\ u_1 & u_4 & -4 & 1 & 0 & \\ u_2 & u_5 & 6 & -4 & 1 & \\ 0 & u_6 & -4 & 6 & -4 & \dots \\ 0 & 0 & 1 & -4 & 6 & \\ 0 & 0 & 0 & 1 & -4 & \\ & & \vdots & & & \end{pmatrix}$$

where $u_0, \dots, u_6$ are unknowns. Note that we choose three values in the first column and four values in the second to be unknown. Larger masks can be used, but larger masks imply that the spread of the basis will also be larger. This is because the basis functions are linear combinations of the Green's functions and the difference mask. Our goal is to set up conditions to solve for the u's. To ensure that the basis functions form a partition-of-unity, we need to solve for an N that annihilates constants on the right-hand side. The reasoning for this condition is simple. Assume the N annihilates constants on the right-hand-side. If we multiply both sides of Eq 2.13 by a column

vector c of 1's, the equation reduces to $N_k\mathbb{B}c = 0$. Since $\mathbb{B}c$ is in the nullspace of N, $\mathbb{B}c$ must be a constant vector.

In addition to partition-of-unity, we also want to construct a set of basis that possesses *linear reproduction*. That is, given samples of a linear function $p_1, \ldots, p_m$, $f(t) = \sum_{i=1}^{m} p_i B_i(t)$ is linear. (Linear reproduction is essential in image deformation and other space deforming tasks [43]). Using the same reasoning as in the partition-of-unity case, we conclude that for the basis to possess linear reproduction, N needs to annihilate linears on the right-hand-side. So the system we need to solve is $N.(c_0 c_a) = 0$, where c_0 is a column vector of 1's. c_1 is column vector with values of the blossom of a linear function over the knots.

The three-variable blossom of a linear function $f(x) = x$ is $g(s_0, s_1, s_2) = (1/3)(s_0 + s_1 + s_2)$. Therefore, c_1 is simply the sum of the knots three-at-a-time divided by three. However, the smallest and largest knot (in one dimensions) are the boundaries of the domain, and we do not have enough information to compute the blossom at these boundary points. This unknown represents one extra degree of freedom in the solution. In practice, we simply reflect the interior knot that is closest to the boundary (e.g. second and second-to-last knot in the sequence) over the boundary knot to compute the blossom of the linear function over the boundary.

In computing the discrete B-spline basis functions centered over a knot set T_0, we consider a dense set of knots T_k where $T_0 \subset T_k$. By solving $N.(c_0 c_1) = 0$ for both the dense and input knot sets, we obtain two difference matrices N_0 and N_k. Both matrices have two vectors in their nullspace by design (the constant and linear blossoms). To solve for $\mathbb{B}$ in Eq. 2.7, we need to eliminate this nullspace by imposing additional conditions on the system. One such condition can be gleaned from Eq. 3.15, which indicates that the sum of the integral of basis functions in the coarse knot space is the preserved in the fine knot space. The integral constraints removes one vector from the nullspace of the difference matrix. To remove the second vector, we impose

the first moments constraint, where the first moments of the basis functions in the coarse knot space are preserved in the fine knot space. More formally, we solve the simultaneous system,

$$\begin{pmatrix} N_k \\ a_k^0 \\ a_k^1 \end{pmatrix} \mathbb{B} = \begin{pmatrix} N_0 \\ a_0^0 \\ a_0^1 \end{pmatrix},$$

where a_k^0 and a_0^0 are the integrals of the basis functions (zero-th moments) for the fine and coarse knot space, and a_k^1 and a_0^1 are the first moments of the basis functions for the fine and coarse knot space. Given this linear system, our next step is to derive the moments constraints a^0 and a^1.

In the unbounded 1D case, a^0 and a^1 are well-known [62]. $a_{(i)}^0$, the integral of the i-th basis, can be written as $(1/4)(t_{i+2} - t_{i-2})$. $a_{(i)}^1$, the first moment of the i-th basis can be written as $(1/20)\sum_{j=i-2}^{i+2} t_j$. For the bounded case, the integral and first moments for knots near the boundary cannot be computed using these standard formulations. Instead, we note that each cubic B-spline basis function is a piece-wise cubic function, and each piece can be written as a cubic Bezier functional curve. Figure 2.5 shows a boundary B-spline basis function (red curve), and the basis function consists of two cubic polynomials, which are separated by the "+" sign in the figure. The integral of the functional Bezier is simply the sum of its coefficients (up to a constant multiple). C^2 continuity of the 1D B-spline imposes conditions on the coefficients of adjacent cubic Bezier functions. Further conditions can be gathered from the property that B-spline basis functions form a partition-of-unity.

For simplicity, we only describe the conditions for the B-spline basis associated with the lower boundary of the domain; the upper boundary has similar derivation. Let the ordered list of knots at the lower boundary be $\{t_0, t_1, t_2, t_3\}$. We further divide each knot interval into quadruple of equally-spaced sub-knots, where $(t_0 = t_{0,0}) < t_{0,1} < t_{0,2} < (t_{0,3} = t_1)$ and $(t_1 = t_{1,0}) < t_{1,1} < t_{1,2} < (t_{1,3} = t_2)$ (see Figure 2.5). Each quadruple of sub-knots are associated with coefficients of the Bezier curves that

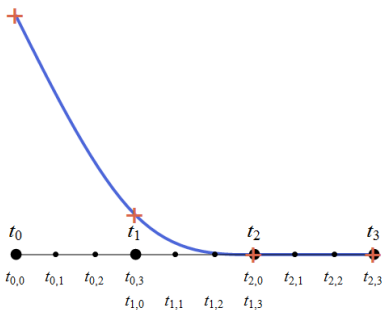


Figure 2.5 : Notation for the knots and sub-knots for the bounded basis. The red curve is a B-spline basis near the boundary. The t_i's are knots near the boundary. The B-spline basis function is composed of two polynomial curves separated by the + mark.

form the B-spline basis. We write the coefficient associated with a sub-knot $t_{i,j}$ as $k_{i,j}$. Consider the basis function centered at t_0, our goal is to derive two sets of Bezier coefficients $k_{0,0}, \ldots, k_{0,3}$ and $k_{1,0}, \ldots, k_{1,3}$. Cubic B-spline basis has certain invariant properties that determine the conditions that the Bezier coefficients must satisfy. We list these properties,

1. The basis functions form a partition-of-unity.
2. Each basis function vanishes outside the bound of two knots away from its center.
3. Each basis function is C^2 at the knots.
4. The second derivative of the basis function centered at t_0 vanishes at t_0 and t_2.

These properties inform the following constraints on the Bezier coefficients for the basis function centered at t_0. The number in the parenthesis corresponds to the properties of the B-spline basis listed above. Let b_{t_0,t_1} and b_{t_1,t_2} be the two Bezier curves over the ranges $[t_0, t_1]$ and $[t_1, t_2]$.

- $k_{0,0} = 1$ (1)

- $k_{1,1} = k_{1,2} = k_{1,3} = 0$ (2 and 4)
- $k_{0,3} = k_{1,0}$ (3)
- $b^{(1)}_{t_0,t_1}(t_1) = b^{(1)}_{t_1,t_2}(t_1)$ (3)
- $b^{(2)}_{t_0,t_1}(t_1) = b^{(2)}_{t_1,t_2}(t_1)$ (3)
- $b^{(2)}_{t_0,t_1}(t_0) = 0$ (4)

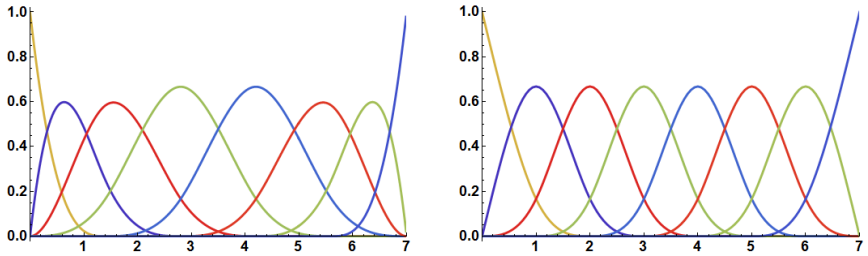
Given these constraints, the unknown Bezier coefficients can be solved by arithmetic manipulation. Similar constraints can be derived for the knot centered at t_1 and at the knots of the upper boundary. Given the Bezier coefficients for the corresponding B-spline basis functions, we can derive the zero-th and first moments of the first two knots (t_0 and t_1) by simply summing the coefficients. To simplify the formulas, we write $\alpha = t_1 - t_0$, $\beta = t_2 - t_1$, and $\gamma = t_3 - t_2$, where α, β, and γ are the length of the first three knot intervals. These formulas are,

$$
\begin{aligned}
a^0_1 &= \frac{3\alpha^2 + 3\alpha\beta + \beta^2}{8\alpha + 4\beta} \\
a^0_2 &= \frac{3\alpha^2 + 4\alpha\beta + \beta^2 + 2\alpha\gamma + \beta\gamma}{8\alpha + 4\beta} \\
a^1_1 &= \frac{1}{20}(2\alpha^2 + 2\alpha\beta + \beta^2) \\
a^1_2 &= \frac{1}{20}(4\alpha^2 + 6\alpha\beta + 2\beta^2 + 4\alpha\gamma + 3\beta\gamma + \gamma^2)
\end{aligned}
$$

Recall a^0_1 and a^0_2 denotes the zero-th moment (integral) for the first and second knot (or t_0 and t_1) in the knot sequence, and a^1_1 and a^1_2 are similarly defined for the first moment.

2.3.2 Analysis of the bounded case

Bounded B-spline basis is typically constructed by repeating knots at the boundary. The number of repetition, or multiplicity, of the knot is related to the order of the spline. For example, a third-degree B- spline would repeat the boundary knots by four



(a) (b)

Figure 2.6 : The bounded B-spline basis functions constructed using knot multiplicity (a). The bounded basis functions constructed using refinement and divided differences (b).

times. The resulting B-splines basis functions possess linear reproduction. Figure 2.6 shows the B-spline basis functions for a set of uniformly spaced knots with knot multiplicity at the boundaries. Note that the peaks of the basis functions do not align with the knots. In contrast, the B-spline basis functions created from our scheme peaks exactly at the knots, which matches the expectation of the construction. In the univariate case, this difference is negligible from a design standpoint because knot spaces can be easily adjusted. In the bivariate case, however, having knot-centered basis functions benefits applications such as image deformation.

2.4 Future work and discussion

Admittedly, our presentation of the differential (or, as labeled in Chapter 3, biharmonic) B-splines is mostly relies heavily on construction and not as much on mathematical derivation. For example, we note that the Green's function is unknown in the bounded case, and therefore we do not have an analytic form for the corresponding biharmonic B-splines. However, B-splines for bounded domains by ways of multiplicity have long been known, and we have not established the connection between the biharmonic B-splines and the traditional bounded B-splines. One important question

that we side-stepped is that whether biharmonic B-splines in the bounded, univariate case is truly piece-wise polynomial. Our intuition and numerical results appear to confirm that assumption in the positive, but we have yet to arrive at a proof.

Another possible future direction is to show that our refinement scheme is convergent. Again, we have numerical results in Chapter 3 that suggests that the scheme does, in fact, converge. However, we still lack the analytical proof of that result. One more interesting area to explore in the unbounded case is the relationship between our refinement equations and traditional knot insertion schemes, such as Lane-Riesenfeld or Bohem's methods [45, 5]. We believe these two approaches to refinement must be equivalent and perhaps by establishing the relation between the refinement equations and the blossom of the spline, we can arrive at such a unifying connection.

In this chapter, we reviewed the construction of the univariate B-spline as a primer for the next chapter. In particular, we studied a derivation of B-splines using solutions to harmonic differential equations. Our differential presentation is equivalent to the classical derivation of B-splines using one-sided power functions and divided difference. We showed that the properties of bump-likeness and locality of B-spline can be derived using the differential equations. We also formulated conditions for computing difference masks for boundary knots that result in linear reproducing basis functions. In the next chapter, we extend our univariate construction into the bivariate domain and show that many of the attractive B-spline properties are retained in the extension.

Chapter 3

Discrete Bi-Laplacians and Biharmonic B-splines

3.1 Motivation

Divided differences and B-splines are two core mathematical methods for modeling with smooth curves. For surfaces, the notions of divided differences and their corresponding splines are less well-established. While mesh modeling methods such as subdivision surfaces are capable of delivering high-quality smooth surfaces, the theory backing these methods lacks the flexibility and elegance of the univariate case. For example, cubic B-splines possess a discrete theory based on divided differences that enables modeling with these splines over fully irregular knot sequences. These methods produce discrete approximations that converge to a corresponding smooth curve in the limit. While some elements of such a theory exist for two-dimensional domains (such as the discrete Laplacian), the core elements of such a theory that generalizes divided differences to irregular meshes is currently unknown.

3.1.1 Divided differences and B-splines

In the univariate case, divided differences and B-splines are intricately woven together. To illustrate this idea, we consider the following construction for cubic B-splines. The starting point is the *Green's function* $\phi_y(x)$ satisfying the differential equation

$$\Delta^2 \phi_y(x) = \delta(x - y) \tag{3.1}$$

where Δ computes the second derivative with respect to x of $\phi_y(x)$ (the univariate analog of the Laplacian) and δ is the Dirac delta function. In our notation, the

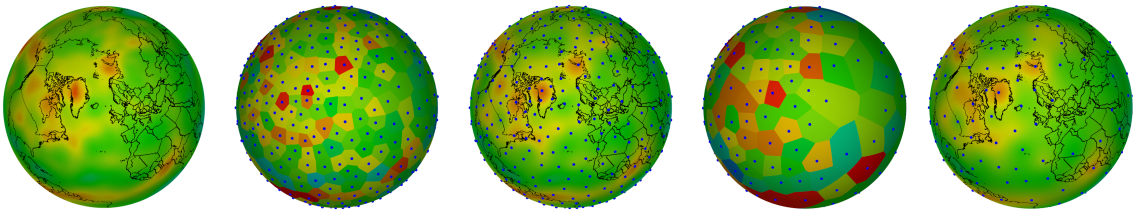


Figure 3.1 : Approximation using biharmonic B-splines on the sphere. Left to right: Atmospheric carbon monoxide concentration (source: *Science on a Sphere*); Dense knot set (350 knots) with associated Voronoi tiles colored by corresponding B-spline coefficients; Dense biharmonic B-spline generated by knots and coefficients; Coarse knot set (150 knots) with associated Voronoi tiles colored by corresponding B-spline coefficients; Coarse biharmonic B-spline generated by knots and coefficients. Note that B-spline coefficients approximate the associated functions due to the local, bump-like shape of biharmonic B-spline functions.

subscript y denotes that the Green's function $\phi_y(x)$ is *centered* at y. This equation has a symmetric solution of the form $\phi_y(x) = \frac{1}{12}|x-y|^3$. Note that, by construction, the Green's function $\phi_y(x)$ is C^2 everywhere even at $x = y$.

Given a set of five knots $t_i = i$ for $i = -2...2$, we construct a cubic B-spline function $B(x)$ as linear combination of translates of this Green's function centered at each knot,

$$B(x) = \sum_i n_i \phi_{t_i}(x). \tag{3.2}$$

Classical B-spline theory [15](Section XI) states that the appropriate choice of localizing coefficients n_i corresponds to the difference of the third divided differences of the first four knots and last four knots. The resulting difference *mask* n_i has the form

$$(1, -4, 6, -4, 1) = (0, -1, 3, -3, 1) - (-1, 3, -3, 1, 0). \tag{3.3}$$

Figure 3.2(left) shows a picture of this construction. Due to the symmetry of the Green's function, this construction for $B(x)$ can be recast in a *differential* form that

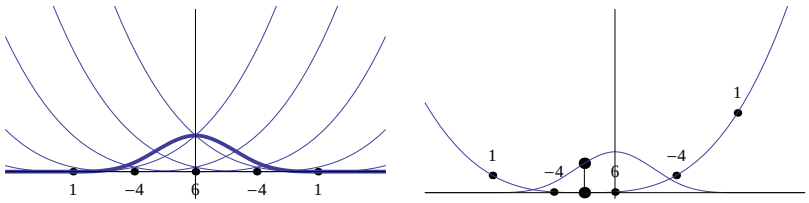


Figure 3.2 : Construction of B-spline function via preconditioning (weighted combination of translates) (left), construction via differential approach (right).

makes the connection between the differential operator Δ and the difference mask n_i more apparent,

$$B(x) = \sum_i n_i \phi_x(t_i). \tag{3.4}$$

In this view, the function $B(x)$ results from applying the mask n_i to samples of the Green's function centered at x. Figure 3.2(right) shows an example of this view of the process. If the mask n_i approximates the action of Δ on the knots t_i in an appropriate manner, Eq. 3.4 is simply a discrete version of Eq. 3.1 and the resulting smooth function $B(x)$ is an approximation to the Dirac delta function. In this view, our fundamental approach is to generalize Eq. 3.1 in interesting ways and develop a recipe for constructing difference masks that approximate Δ. If this recipe is well-chosen, the resulting functions $B(x)$ will be well-behaved analogs of univariate B-splines.

3.1.2 Related work

In the regular case, building higher order analogs of the discrete Laplacians is easy and the uniform splines associated with these operators are well-studied [67, 83, 79]. Our work focuses on the fully irregular case and draws its primary inspiration from the preconditioning methods for scattered data interpolation with radial basis functions of [19, 10, 9, 53]. Direct methods for interpolation with radial basis functions often involve constructing an interpolation matrix whose condition number is poor

due to the unbounded shape of radial basis functions. One classical technique for improving this condition number is to left-multiply the interpolation matrix by a *preconditioning* matrix N. [19] gave such a construction for a class of preconditioning matrices N formed by discretizing the differential operator associated with the radial basis function and observed that applying this preconditioner to the interpolation problem could be viewed as a change of basis (via equation 3.2) into a new set of globally defined, but usually localized "bell-shaped" basis functions.

Our work advances this idea by establishing conditions on the preconditioner to guarantee the resulting combinations of radial basis function are both provably localized and form a partition of unity. (Classical scattered data methods force low-order polynomial reproduction via auxiliary fitting constraints). Our work also generalizes this idea to smooth manifolds in 3D, where only a limited amount of progress has been made. (See [28] for a discussion of scattered data interpolation on the sphere).

Many papers have explored the properties of various versions of the discrete Laplacian (see [82] for an overview). More recent mesh processing work has investigated the biharmonic case for various applications [78, 85, 48, 40]. For these schemes, the typical approach has been to use an iterated version of the discrete Laplacian. While the iterated discrete Laplacian has the advantage of being simple, we will demonstrate that this mask has significant drawbacks in comparison to our discrete bi-Laplacian.

Other methods from discrete differential geometry have considered discretization of other differential operators [16]. However, we know of no spline-based formulations or progressive refinement strategies for such methods. More generally, the problem of solving variants of the biharmonic equation using the finite difference method is well-studied [75]. However, in most formulations, the problem is posed as a pure boundary value problem. In our formulation, we imposed the boundary conditions at the knots of the coarse domain so as to mimic the differential formulation of univariate B-splines.

Other methods have focused on building splines/interpolants as smooth piecewise polynomials over irregular planar geometries [14]. The most successful approach due to [61] generates smooth splines on irregular triangulations. However, this method does not possess a discrete progressive refinement method and has no clear generalization to curved domains. Natural neighbor methods based on Voronoi diagrams [73, 38] have also been used to interpolate scattered data, and later works extended natural neighbors to surfaces [7]. Again, these methods also lack a discrete progressive refinement method (and produce functions that are strictly C^0 at the data sites).

On curved domains, [57] studied versions of harmonic and biharmonic Bezier surfaces. More generally, methods from differential geometry (too numerous to cite) have been proposed to build splines on manifolds via either a piecewise patch structure or overlapping charts. We observe that, while our method does compute local parameterizations in generating the discrete bi-Laplacian mask on curved domains, these parameterizations are easy to construct and have no compatibility conditions between adjacent parameterizations.

3.1.3 Contributions

We analyze the structure of the discrete Laplacian when used as a preconditioner for harmonic Green's functions and expose two important structural properties of this mask. One structural property determines the localization of the preconditioned Green's functions while the other property ensures that the preconditioned functions form a partition of unity. Our fundamental contribution is to leverage these observations and develop a general recipe for generating difference masks associated with a partial differential equation. These masks have the property that, when used as a preconditioner on translates of the associated Green's function $\phi_y(x)$, the resulting preconditioned functions $B_j(x)$ are localized and form a partition of unity.

In the case of the biharmonic equation, we develop a new discrete bi-Laplacian and

show that its corresponding biharmonic B-splines are localized and form a partition of unity. We then develop generalizations of both the discrete Laplacian and the discrete bi-Laplacian for curved domains and show that their associated B-splines are again well-behaved. We conclude by proposing a discrete progressive refinement scheme for these splines that enables discrete modeling with these splines in the irregular case.

3.2 The discrete Laplacian

In the univariate case, the difference mask of Eq. 3.3 was a discrete approximation to the continuous fourth order derivative operator on a uniform knot sequence. In the bivariate case, the discrete Laplacian approximates the continuous Laplacian Δ (the sum of the second derivatives of each argument of a bivariate function). In this section, our goal is to mirror the univariate construction and derive a set of bivariate basis functions constructed from translates of the Green's functions for Δ that behave in a manner similar to the univariate B-splines.

The starting point of our construction is the *harmonic* equation

$$\Delta\phi_y(x) = \delta(x - y) \tag{3.5}$$

where x and y are points in the plane. One symmetric solution to this equation is the Green's function $\phi_y(x) = \frac{1}{2\pi}\log(|x - y|)$. The normalizing factor $\frac{1}{2\pi}$ is notable since our goal will be to build a set of localized functions that form a partition of unity. Given a region Ω in the plane, applying Green's theorem [26] (pp. 345) yields that

$$\mathbb{Z}_{x\in\Omega}\Delta\phi_y(x)dA = \mathbb{Z}_{x\in\partial\Omega}\frac{\partial\phi_y(x)}{\partial\nu}ds \tag{3.6}$$

where ν is the outward unit normal to the boundary $\partial\Omega$ of Ω. If we choose Ω to be the unit disk centered at y, the left hand side of Eq. 3.6 is exactly one. On the other hand, the derivative of $\log(x)$ with respect to x is $\frac{1}{x}$ which evaluates to one at $x = 1$. Therefore, we must normalize the Green's function by a factor of $\frac{1}{2\pi}$ to account for the circumference of the unit disk Ω in this boundary integral.

3.2.1 The construction as a boundary sum

In constructing the discrete Laplacian, we treat the knots t_i as points in the parameter plane x. In the case of univariate splines, the knots were ordered in ascending value and then clustered to form discrete difference masks using consecutive sequences of knots. In two dimensions, we cluster the knots using nearest neighbors. For each knot t_i, we define the set of all points closest to t_i as the *Voronoi* tile $\mathcal{V}_i$. If the Voronoi tiles $\mathcal{V}_i$ and $\mathcal{V}_j$ share a common *Voronoi edge* v_{ij}, we construct a corresponding *Delaunay edge* e_{ij} between these two knots. These Delaunay edges induce a *Delaunay triangulation* T of the knots t_i. Note that, by construction, a Voronoi edge v_{ij} is perpendicular to its corresponding Delaunay edge e_{ij}.

We now form a *preconditioning matrix* N whose columns corresponds to the standard discrete Laplacian applied to one-ring of each knot in this triangulation. In the discrete Hodge star formulation [16], the non-zero entries of each column (the *mask* associated with the column) have a simple expression in terms of the lengths of the Delaunay edges and their corresponding dual Voronoi edges. Given a Delaunay edge e_{ij}, the corresponding entry n_{ij} of N is exactly $|v_{ij}| \,/\, |e_{ij}|$. The diagonal entries of N are the negative of the sum of the off-diagonal entries in their corresponding column to ensure that sum of the entries in each column is zero.

Observe that the discrete Laplacian mask centered at t_j consists of the sum of weighted divided differences along the edges e_{ij} incident to the central knot t_j. Each divided difference approximates a directional derivative normal to the boundary of the shaded Voronoi tile $\mathcal{V}_j$ as shown in Figure 3.3(left). Since the discrete Laplacian weights each divided difference by the length of its corresponding dual Voronoi edge, the resulting sum approximates a boundary integral of the form,

$$\sum_{v_{ij} \in \partial \mathcal{V}_j} \frac{|v_{ij}|}{|e_{ij}|} (p(t_i) - p(t_j)) \simeq \mathbb{Z}_{x \in \partial \mathcal{V}_j} \frac{\partial p(x)}{\partial \nu} ds. \tag{3.7}$$

Similarly, the sum of several discrete Laplacians (taken over their corresponding

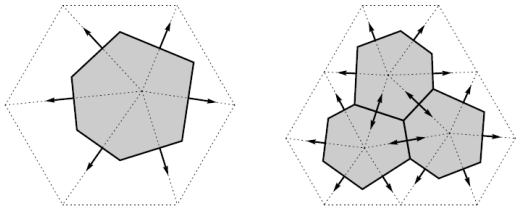


Figure 3.3 : Discrete Laplacian as a boundary sum on Voronoi tile (left), sums of discrete Laplacians as a boundary sum (right)

shaded Voronoi tiles as shown in Figure 3.3(right)) reduces to a boundary sum over the shaded region since the weighted differences along interior Delaunay edges cancel.

In cases where $p(x)$ is a low degree polynomial, discrete approximations such as this one may return the exact value of the continuous integral. In such cases, we say that the discretization is *precise*. The discrete Laplacian has linear precision since both sides of Eq. 3.7 reduce to zero for linear functions. While the discrete Laplacian is not precise on all quadratics, we observe that it is precise on the quadratic function $q(x) = \frac{1}{4}|x|^2$. Since $\Delta q(x)$ is identically one, the righthand side of Eq. 3.7 reduces to exactly the area of Ω. On the other hand, the row vector q whose jth entry is $q(t_j)$ satisfies $qN = a$ where a is a row vector whose jth entry is the area of $\mathcal{V}_j$. (This observation follows easily if we translate the knot set such that t_j lies at the origin).

3.2.2 Harmonic B-splines and their properties

We next investigate the effect of using the discrete Laplacian as a preconditioner for the Green's functions associated with the Laplacian. Given the discrete Laplacian masks n_{ij} and the Green's function $\phi_y(x)$, we define the *harmonic B-spline* functions to be linear combinations of translates of the Green's function $\phi_y(x)$ of the form

$$B_j(x) = \sum_i n_{ij}\phi_{t_i}(x).$$

Just as in the univariate case, this preconditioned definition of harmonic B-splines has an equivalent differential form $\sum_i n_{ij}\phi_x(t_i)$. In this form, the function $B_j(x)$ is computed by sampling the Green's function centered at x via the discrete Laplacian centered at t_j. As the center x of the Green's function varies, the corresponding discrete differences form $B_j(x)$. In its differential form, the definition of the harmonic B-spline is simply a discrete version of Eq. 3.5 with $B_j(x)$ approximating $\delta(x - t_j)$ in some sense.

Figure 3.4 shows two harmonic basis functions built using this differential construction. The left-most column shows the one-rings of two knots, their discrete Laplacian masks and a particular parameter value x (plotted as a large dot). The middle column shows plots of the Green's function centered at x and sampled at the knots of the one-ring (small red spheres). The right-most column shows plots of harmonic basis functions whose height at x (the small sphere) is the sum of the mask values in the first column times the heights in the second column. Note that both harmonic B-splines have poles at their centers and adjacent knots since the harmonic Green's function possesses a pole at its center. In spite of these poles, harmonic B-spline functions possess three important properties that are a direct consequence of the structure of the discrete Laplacian as a boundary operator.

Localized As opposed to the univariate case, harmonic B-spline functions are globally supported. However, they decay quickly towards zero as x varies away from their central knot. This property is a consequence of the linear precision of the discrete Laplacian as a boundary operator.

Theorem: Given a preconditioner N that is precise to degree d, fix $x \neq 0$ and let the scalar r vary towards infinity. The sequences of values $B_j(rx)$ decays towards zero at the rate $O(\frac{1}{r^{d+1}})$.

Proof: Consider the effect of dilating the knot sets via $\frac{1}{r}t_i$ as $r \to \infty$ in both the

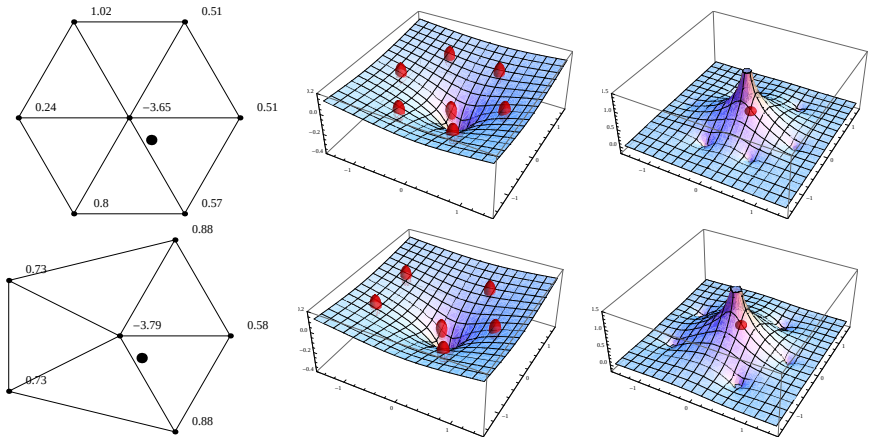


Figure 3.4 : The differential construction for two harmonic B-spline functions, top is regular 3-direction, bottom is irregular

preconditioned and differential view. In the preconditioned view of B_j,

$$\sum_i n_{ij}\phi_{t_i/r}(x) = \sum_i n_{ij}\phi_{t_i}(rx) = B_j(rx)$$

since $\phi_{t_i/r}(x) = \phi_{t_i}(rx) - \frac{1}{2\pi}\log(r)$. (The weights n_{ij} annihilate the constant $\frac{1}{2\pi}\log(r)$ when substituted into the above sum). In the differential view, the sequence of values $\sum_i n_{ij}\phi_x(r^{-1}t_i)$ converges to zero at rate of $O(\frac{1}{r^{d+1}})$ by Taylor's theorem since the difference masks n_{ij} are precise to order d. Thus, $B_j(rx)$ converges to zero at a rate of $O(\frac{1}{r^{d+1}})$ as $r \to \infty$. QED

Given that the discrete Laplacian is precise on linear functions, the basis functions $B_j(x)$ decay radially towards zero from their central knot t_j at the rate $O(\frac{1}{r^2})$. This rate of decay allows the functions to be integrated radially from their central knot t_j. Based on numerical experiments, we conjecture that this integral is exactly the area a_j of the Voronoi tile $\mathcal{V}_j$. If q is a row vector whose jth entry is $\frac{1}{4}|t_j|^2$, the integrals of the functions $B_j(x)$ are exactly $qN = a$.

Bump-like We claim that $B_j(x)$ has a bump-like shape that approximates the characteristic function for the tile $\mathcal{V}_j$. In particular, we note that $\sum_i n_{ij}\phi_x(t_i)$ corresponds

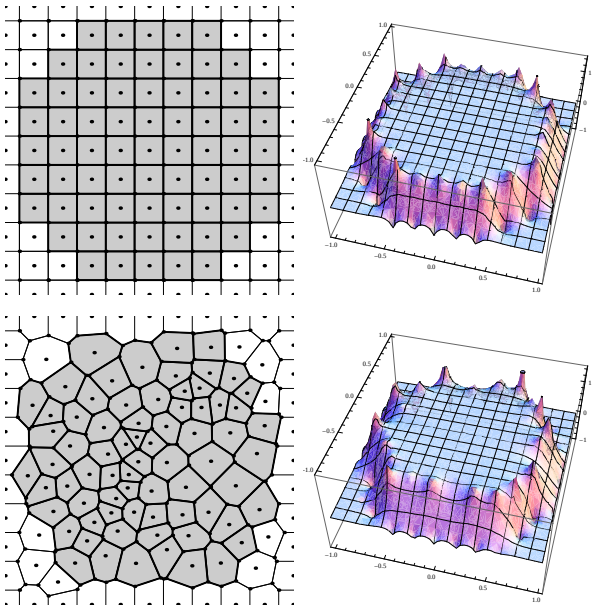


Figure 3.5 : Sums of harmonic B-splines whose centers lie in a fixed disk for regular and irregular knots sets

to a boundary sum that approximates the boundary integral from equation 3.6,

$$\sum_{v_{ij}\in\partial\mathcal{V}_j} \frac{|v_{ij}|}{|e_{ij}|}(\phi_x(t_i) - \phi_x(t_j)) \simeq \mathbb{Z}_{y\in\partial\mathcal{V}_j} \frac{\partial\phi_x(y)}{\partial\nu} ds. \tag{3.8}$$

This boundary integral is equivalent to the area integral $\mathbb{Z}_{y\in\mathcal{V}_j}\Delta\phi_x(y)dA$ which reduces to the characteristic function for $\mathcal{V}_j$.

Partition of unity Figure 3.5 shows two examples of harmonic B-splines, one for a regular two-direction grid, one for an irregular grid. In both examples, we have taken the sum of harmonic B-spline functions whose central knots lie in a fixed disk. Two observation are apparent from these examples. First, both harmonic B-splines have poles only along the boundary of the disk and not in their interior. Second, these harmonic B-splines approximate the constant function one on their interiors. Again, the shape of these harmonic B-splines is a direct consequence of the structure of the discrete Laplacian as a boundary operator.

To explain these observations, we introduce a more compact vector/matrix notation. Let $B(x)$ and $\Phi(x)$ be row vectors whose jth entries are $B_j(x)$ and $\phi_{t_j}(x)$, respectively. In matrix/vector form, these row vectors are related by the preconditioning matrix N via $\mathbb{B}(x) = \Phi(x)N$. Returning to the examples of Figure 3.5, let ω be a column vector whose jth entry is 1 if $|t_j| \leq 1$ and 0 otherwise. Similar, let Ω be the union of all Voronoi tiles whose centers lie in the unit disk. We note that the function $\mathbb{B}(x)\omega = \Phi(x)N\omega$ approximates the characteristic function for Ω (as argued for a single Voronoi tile above) since the edge differences in $N\omega$ corresponding to adjacent tiles in Ω cancel (as shown in Figure 3.3(right)).

Unfortunately, no finite combination of the basis functions form an exact partition of unity since the basis function are not local. However, as more basis functions are included in the sum, the resulting functions converge to the unit function (as shown in the appendix):

Partition of unity theorem: Let T be an infinite knot set T whose associated Delaunay and Voronoi edges have a bounded maximal length. If ω_k is a column vector whose jth entry is 1 if $|t_j| \leq 2^k$ and 0 otherwise, the values $\mathbb{B}(0)\omega_k$ converge to one as $k \to \infty$.

3.3 The discrete bi-Laplacian

In this section, we consider the standard generalization of the discrete Laplacian to the biharmonic case and propose a new alternative. The starting point for our planar construction is the biharmonic equation of the form $\Delta^2\phi_y(x) = \delta(x-y)$ where Δ^2 is the iterated Laplacian. In this case, one symmetric solution to this equation has the form $\phi_y(x) = \frac{1}{8\pi}|x-y|^2 \log(|x-y|)$. As opposed to the harmonic Green's function which is C^{-1} at $x = y$, this biharmonic Green's function is C^1 (but not C^2) at $x = y$.

Again, the normalizing constant of $\frac{1}{8\pi}$ arises from applying Green's theorem to the

biharmonic equation,

$$\mathbb{Z}_{x\in\Omega}\Delta^2\phi_y(x)dA = \mathbb{Z}_{x\in\partial\Omega}\frac{\partial(\Delta\phi_y(x))}{\partial\nu}ds. \tag{3.9}$$

This equation is critical in guiding our construction of an appropriate discrete bi-Laplacian. As in the previous section, we construct a preconditioning matrix whose columns mimic the action of the boundary integral on the right hand side of this equation.

3.3.1 Weaknesses of the iterated discrete Laplacian

Our goal is to replicate the structural properties of the discrete Laplacian and form an analogous discrete bi-Laplacian for the biharmonic equation. If we let N_* denote the matrix version of the discrete Laplacian, one choice for this discretization is the *iterated discrete Laplacian*, $N_*A^{-1}N_*$ where A is usually a diagonal matrix whose entries are the areas of the Voronoi tiles $\mathcal{V}_j$. Due to its simplicity, the iterated discrete Laplacian is a popular method for solving the biharmonic Eq. [40] (where it is referred to as a mass-lumped, mixed finite element scheme).

The iterated discrete Laplacian is a symmetric matrix that acts as an excellent preconditioner on regular grids (both two-direction and three-direction). However, on irregular grids, the performance of the iterated Laplacian as a preconditioner is more questionable. In particular, preconditioning with the discrete iterated Laplacian yields biharmonic functions that are neither localized nor form a partition of unity.

Figure 3.6 shows three irregular triangulations (left column) and three biharmonic functions (middle column) generated via preconditioning with the discrete iterated Laplacian. Note that the basis functions in the second and third rows are not localized. Figure 3.8 (left) shows the sum of one hundred biharmonic functions whose central knots lie in a unit disk. Note that the sum is not converging to the constant function one. To understand these issues, we first state a localization result similar to that for

the harmonic case.

Localization theorem: Given a biharmonic preconditioner that is precise on Δ^2 to degree d, the preconditioned functions $B_j(rx)$ decay towards zero at a rate of $O(\frac{1}{r^{d-1}})$ as $r \to \infty$.

The proof of this theorem exactly follows that of the harmonic case with the exception that the biharmonic Green's functions satisfy a scaling relation that introduces an extra factor of r^2, explaining the reduced rate of decay. This theorem points towards the problematic issue. Unfortunately, the iterated discrete Laplacian only has linear precision on irregular grids and, as a result, defines functions via preconditioning that are not guaranteed to be localized.

Lack of precision also explains the partition of unity issue. Note that the discrete Laplacian N_* can be factored in two simpler matrices via $N_* = DB$. The columns of the matrix D compute divided differences on the edges of T while the columns of B weigh these differences by the lengths of their associated Voronoi edges. In terms of Eq. 3.6, the matrix D computes discrete approximations to the normal derivative $\frac{\partial(\phi_y(x))}{\partial \nu}$. In the biharmonic case, the matrix D should now compute discrete approximations to the normal derivative of the Laplacian $\frac{\partial(\Delta\phi_y(x))}{\partial \nu}$ as in Eq. 3.9. Unfortunately, the approximation generated by the iterated discrete Laplacian $D_* = N_* A^{-1} D$ has only linear precision on irregular grids.

3.3.2 Construction on planar domains

Our solution to this problem is to replace the discrete iterated Laplacian matrix D_* by a new difference matrix D whose columns approximates $\frac{\partial(\Delta\phi_y(x))}{\partial \nu}$ with cubic precision on the edges of T. If the difference mask associated with each column of D is supported over the set r_{ij} of all knots that lie in the intersection of the two-rings of the knots t_i and t_j, the columns of the resulting preconditioner $N = DB$ (where

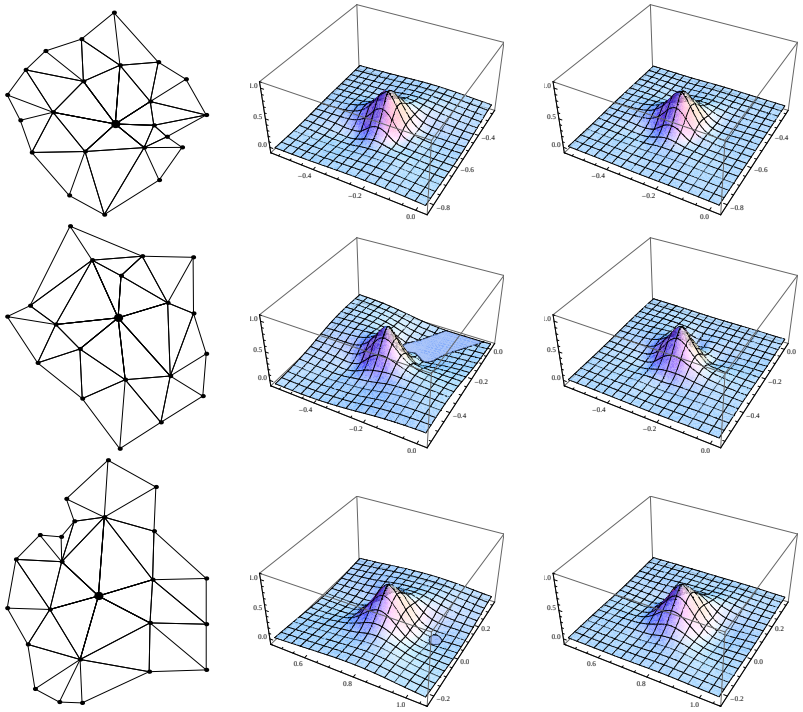


Figure 3.6 : Bi-harmonic functions defined over three irregular triangulations (left column), constructed using the discrete iterated Laplacian (middle column) and discrete bi-Laplacian (right column).

B is the boundary weight matrix for harmonics) are supported over the two-rings of the knots.

Construction Given an edge e_{ij}, its mask d is the solution to a linear system involving a Vandermonde matrix M that has one column for each knot in r_{ij} and one row for each bivariate monomial of degree three or less. To simplify our notation for the monomials used in constructing M, we let α be a multi-index of the form $\alpha = (\alpha_1, \alpha_2)$ with $\alpha_i \geq 0$. We define $t^\alpha = (t)_1^{\alpha_1}(t)_2^{\alpha_2}$ and $|\alpha| = \alpha_1 + \alpha_2$. Now, the (α, k) entry of the Vandermonde matrix M is simply t_k^α where $t_k \in r_{ij}$ and $|\alpha| \leq 3$.

To solve for the new mask d, we solve an equation of the form $Md = c$ where c is a column vector whose αth entry is $\frac{\partial(\Delta x^\alpha)}{\partial \nu_{ij}}$ and ν_{ij} is the unit vector in the direction of the edge e_{ij}. In practice, this system is almost always under-determined so we prefer a mask that satisfies this equation while approximating the third difference mask d_* generated by the iterated discrete Laplacian. Specifically, we seek a mask d that satisfies $Md = c$ and that minimizes $\|d - d_*\|_2$. Such a solution d is easily computed using the pseudo-inverse M^+ of M,

$$M(d - d_*) = c - Md_* \tag{3.10}$$

$$d = M^+(c - Md_*) + d_*. \tag{3.11}$$

These masks d form the columns of a new third difference matrix D that has cubic precision. Taking the product of D with the harmonic boundary matrix B yields a new preconditioner $N = DB$ that also has cubic precision. The columns of this matrix N (supported over the two-rings of each knot) form the *discrete bi-Laplacian* associated with each knot.

Biharmonic B-splines Given this preconditioner N, the *biharmonic B-spline* functions have the form $\mathbb{B}(x) = \Phi(x)N$. Since both D and N have cubic precision, the resulting functions $\mathbb{B}(x)$ are localized and decay to zero at rate $O(\frac{1}{r^2})$. The right column of Figure 3.6 shows the effect of preconditioning using the discrete bi-Laplacian

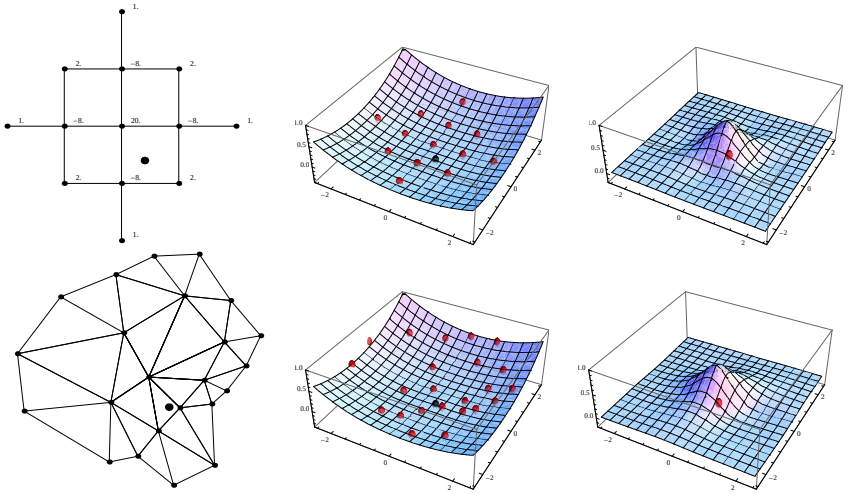


Figure 3.7 : The differential construction for two biharmonic B-spline functions, top row is a regular mesh, bottom row is irregular.

in the irregular case. Note that the resulting functions are now localized. Figure 3.7 shows two examples of the differential constructions of biharmonic B-spline functions for regular and irregular meshes.

Due to cubic precision of D, the biharmonic B-splines functions form a partition of unity based on an argument almost identical to that of the harmonics case. Figure 3.8 shows a comparison of the sums of 100 biharmonic basis functions taken over an irregular triangulation of a square region. Note that the sum based on the discrete iterated Laplacian (left) is not converging to one, while the sum based on the discrete bi-Laplacian (right) is converging to one on the interior of the square region.

In many applications such as functional approximation, the ability to reproduce higher degree polynomials is important. Based on numerical experiments, we believe that harmonic B-splines reproduce linear functions. On the other hand, we have simple examples of irregular knot sets in which biharmonic B-splines appear not to reproduce linear functions. However, we suspect that it is possible to modify our construction of the discrete bi-Laplacian to reproduce linear functions given that it has several degrees

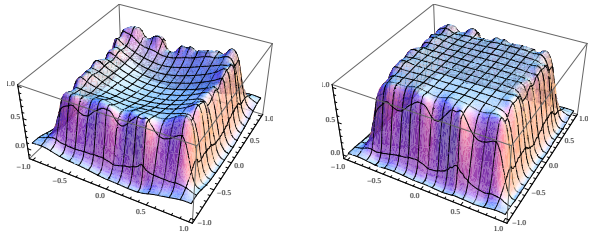


Figure 3.8 : Sums of biharmonic basis functions computed over the same irregular triangulation using the iterated discrete Laplacian (left) and discrete bi-Laplacian (right)

of freedom. We also note that biharmonic B-splines possess the approximation power of thin-plate splines which, as shown in [3], is much better than the $O(h)$ bound predicted by only constant reproduction.

Quadratic decay also makes integration of these biharmonic basis functions feasible. Based on numerical examples, we conjecture that the integral of these basis functions (when computed radially from their central knot) is exactly qN where q is a column vector whose jth entry is $\frac{1}{64}\left|t_j\right|^4$. (The factor of $\frac{1}{64}$ arises from the fact that $\Delta^2(\frac{1}{64}|x|^4)$ is exactly one).

In general, biharmonic B-splines of the form $\mathbb{B}(x)p$ approximate the value of their coefficient p_j at the corresponding knot t_j due to these properties. While biharmonic B-splines do not strictly interpolate, forcing interpolation using biharmonic B-splines is easier since the resulting interpolation problem is well-conditioned and constant reproduction follows without the need for auxiliary constraints.

3.3.3 Construction on curve domains

As in the planar case, the starting point for our construction on a curved compact domain $\mathcal{M}$ is a harmonic/biharmonic equation whose solution is a Green's function

$$\Delta_B^k \phi_y(x) = \delta(x-y) - \frac{1}{area(\mathcal{M})} \tag{3.12}$$

where $x, y \in \mathcal{M}$ and $k = 1$ or 2. (The addition of the constant term to the left hand side is necessary to ensure the existence of a solution $\phi_y(x)$ in the compact case). Here, the Laplace-Beltrami operator Δ_B is a generalization of the standard planar Laplacian Δ to a curve domain (see the appendix for a definition and details).

The discrete Laplacian Constructing a generalization of the discrete Laplacian on a curved domain is straightforward. Given a set of knots $t_i \in \mathcal{M}$, we construct the Voronoi tiles $\mathcal{V}_i$ on $\mathcal{M}$ and let e_{ij} and v_{ij} denote the Delaunay and Voronoi edges shared by two adjacent knots t_i and t_j, respectively. For each edge e_{ij} in the Delaunay triangulation, we let $n_{ij} = |v_{ij}| \,/\, |e_{ij}|$ where $|v_{ij}|$ and $|e_{ij}|$ are *geodesic* distances computed on $\mathcal{M}$. (In degenerate cases where two Voronoi tiles touch along multiple distinct Voronoi edges, we allow multiple Delaunay edges and sum their corresponding contributions).

Given this preconditioner N, we construct the *harmonic B-splines* functions via

$$B_j(x) = a_j + \sum_i n_{ij} \phi_{t_i}(x). \tag{3.13}$$

where $a_j = area(\mathcal{V}_j)/area(\mathcal{M})$. Note that the harmonic B-spline functions trivially form a partition of unity since the contribution of various Green's functions cancel and cause their sum to reduce to the sum of the areas of the Voronoi tiles divided by area of $\mathcal{M}$.

For compact manifolds $\mathcal{M}$ of sufficient smoothness, there always exists a symmetric solution $\phi_y(x)$ to Eq. 3.12 (see [41]). Due to this symmetry, Eq. 3.13 has a differential interpretation similar to that of the planar case. The left column of Figure

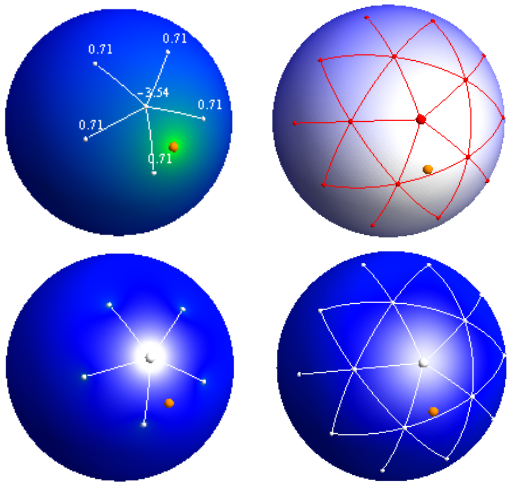


Figure 3.9 : The differential construction for harmonic (left) and biharmonic (right) B-splines. The top row show temperature plots of Green's functions. The bottom row show temperature plots of a B-spline function.

3.9 illustrates this case for the sphere where $\phi_y(x) = \frac{1}{4\pi}\log(\frac{1}{2}(1 - x \cdot y))$. The upper left image shows a temperature plot (green negative, blue zero, white positive) of the Green's function ϕ_x centered at x (the orange dot). A knot t_j and its five neighboring knots are shown in white. The lower right image show a temperature plot of the harmonic B-spline function $B_j(x)$. The value of B_j at x is the sum of values of the Green's function ϕ_x taken at the knots t_i and weighted by their corresponding discrete Laplacian coefficients n_{ij} (plus the normalized area of its Voronoi tile).

In this differential view, these weighted edge differences approximate the boundary integral of Eq. 3.8 taken over the Voronoi tile $\mathcal{V}_j$ on $\mathcal{M}$. This boundary integral has an equivalent area integral (see [26], pp. 363) of the same form as Eq. 3.6 with the sole exception that Δ is replaced by Δ_B.

$$\mathbb{Z}_{y\in\partial\mathcal{V}_j}\frac{\partial\phi_x(y)}{\partial\nu}ds = \mathbb{Z}_{y\in\mathcal{V}_j}\Delta_B\phi_x(y)dA.$$

By the definition of $\phi_y(x)$, this area integral reduces to $1 - a_j$ if $x \in \mathcal{V}_j$ and $-a_j$ otherwise. Since the basis functions $B_j(x)$ include an additive term a_j, these basis

functions approximate the characteristic functions of their Voronoi tiles $\mathcal{V}_j$. Figure 3.10 shows three more examples of harmonic B-spline functions on the sphere, taken over more irregular knot configurations. Note that, as in the planar case, the functions have a bump-like shape and are smooth everywhere but at their knots (where they have poles).

In designing the discrete Laplacian for curved domains, we chose geodesic distances (as opposed to Euclidean distances) to achieve a more accurate discrete approximation of the right hand side of Eq. 3.8. This choice ensures better localization of the resulting harmonic B-spline functions. While these functions do not obey any known asymptotic localization bounds on the compact domain $\mathcal{M}$, their localization increases as the spacing of the knots on $\mathcal{M}$ decreases due to Taylor's theorem. Figure 3.10(bottom) shows histograms of the values of these basis functions. Note that spikes at the origin indicate localization.

The discrete bi-Laplacian Our construction for the discrete bi-Laplacian on curved domains follows the planar approach with a modification to incorporate the effect of the Laplace-Beltrami operator as suggested in [84]. For each edge on e_{ij}, we construct a discrete third difference mask d supported over a neighborhood of the edge e_{ij} that mimics the continuous third derivative operator $\frac{\partial(\Delta_B \phi_y(x)))}{\partial \nu_{ij}}$. Our construction consists of the following steps:

- Construct a local planar parametrization s_k for the knots $t_k \in r_{ij}$ that respect the shape of $\mathcal{M}$ in the neighborhood of e_{ij} (parametrization details in Section 3.4.3).
- Form the Vandermonde matrix M whose (k, α)th entry is ${s_k}^{\alpha}$ where $t_k \in r_{ij}$ (the intersection of the two rings of t_i and t_j) and $|\alpha| \leq 3$.
- Construct a cubic parametric surface $\mu(s) = \Sigma_{|\alpha| \leq 3} \mu_\alpha s^\alpha$ such that $\mu(s_k)$ approximates t_k.

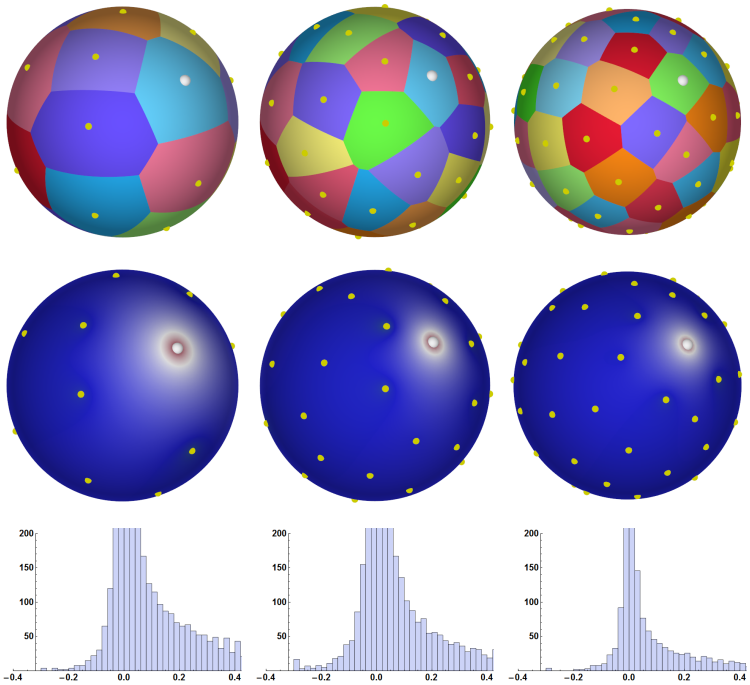


Figure 3.10 : Voronoi tiles on the sphere (top). Three harmonic B-splines on the sphere (middle), corresponding histograms of their values (bottom). The histograms ranges from $-.4$ to $.4$.

- Construct a right hand side c whose αth entry is $\frac{\partial(\Delta_B s^\alpha)}{\partial(\nu_{ij})}$ where Δ_B is computed with respect to the (vector) coefficients μ_α of the local parametrization $\mu(s)$ (see appendix for details). Normalize the result by $\frac{1}{\nu_{ij}\cdot\nabla(\mu(s))}$.
- Solve for the mask d using Eq. 3.11 where d_* is the third difference generated from the iterated discrete Laplacian on $\mathcal{M}$.

Spherical biharmonic B-splines As an example, we specialize this construction to the unit sphere. For each edge e_{ij} of this triangulation (an arc of a great circle), we compute the midpoint of this edge and orthogonally project the knots t_k in r_{ij} onto the plane tangent to the sphere at this midpoint such that the projected edge e_{ij} lies on the s_1-axis. In this tangent plane, we then project these knots radially from this midpoint so as to preserve their geodesic distances on the sphere to form the local parametrization s_k.

In computing the Laplace-Beltrami operator, we observe that the cubic surface $\mu(s_1, s_2)$ of the form

$$\left\{ s_1 - \frac{1}{6}\left(s_1 + s_2\right) s_1^2, s_2 - \frac{1}{6}\left(s_1 + s_2\right) s_2^2, -\frac{1}{2}\left(s_1^2 + s_2^2\right) \right\}$$

is a reasonable local approximation to the sphere and is independent of the position of the tangent plane. Next we compute the coefficients c_α used in Eq. 3.11 as $\frac{\partial(\Delta_B s^\alpha)}{\partial s_1}$ using this local parametrization $\mu(s)$. The resulting coefficient are of the form $c_{(0,1)} = \frac{1}{3}$, $c_{(1,0)} = -1$, $c_{(1,2)} = 2$ and $c_{(3,0)} = 6$ with the remaining coefficients $c_\alpha = 0$.

Figure 3.11 show examples of three biharmonic B-spline functions plotted as temperature maps using the biharmonic Green's function [28] (pp. 186)

$$\phi_y(x) = \frac{1}{4\pi}\text{polylog}(2, \frac{1}{2}(1 + x \cdot y)).$$

These functions share the same properties as harmonic B-splines with the critical advantage that these functions are smooth at their knots.

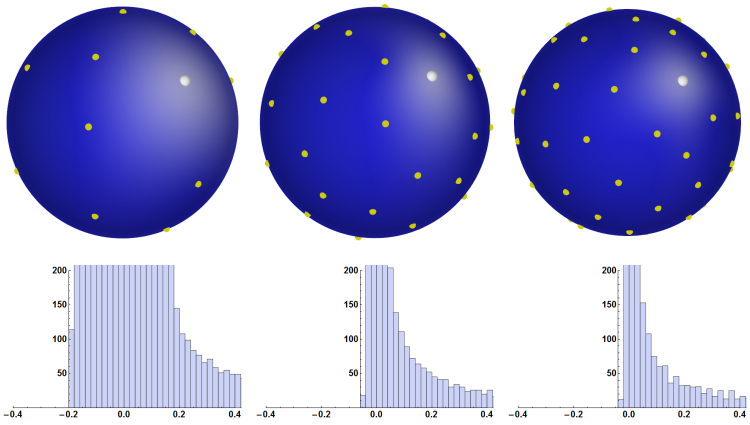


Figure 3.11 : Three biharmonic B-splines on the sphere (top), histograms of their values (bottom).

3.4 Mesh refinement via discrete bi-Laplacians

Univariate B-splines possess an elegant discrete refinement scheme known as *knot insertion* [34]. Given a set of coefficient p_0 defined on a knot sequence T_0, knot insertion computes a new set of coefficients p_1 defined on a denser knot sequence T_1 such that p_0 and p_1 represent the same B-spline. In the univariate case, knot insertion enables fully discrete methods for manipulating B-splines without reference to their underlying piecewise polynomial structure due to the convergence of the dense coefficients to the spline. Knot insertion also allows local editing of the spline near a newly inserted knot.

In this section, we develop a discrete refinement scheme of biharmonic B-splines that generalizes knot insertion for univariate B-splines and enables a fully discrete approach to modeling and manipulating biharmonic B-splines. By removing the need for an analytic form of the Green's function, this scheme extends our construction for biharmonic B-splines to any smooth compact domain.

3.4.1 The direct refinement equations

The starting point for our scheme is a nested sequence of knot sets $T_0 \subset T_1 \subset ... \subset T_k$ all lying on a smooth compact manifold $\mathcal{M}$. Given a set of scalar values p_0 associated with the knots T_0, we can define an associated biharmonic B-spline via $p_0(x) = \mathbb{B}_0(x)p_0$. Now, our goal is to develop a refinement scheme that generates a new set of coefficients p_k defined on T_k such that the $p_0(x) = \mathbb{B}_k(x)p_k$.

Recall that the biharmonic B-splines functions $\mathbb{B}_k(x)$ can be written in vector form as $a_k + \Phi_k(x)N_k$ where $\Phi_k(x)$ is a row vector of translated Green's functions, N_k is the preconditioner formed by the discrete bi-Laplacians and a_k is a row vector of integrals (that corresponds to ratio of the areas of the Voronoi tiles on T_k to the area of $\mathcal{M}$). Since $T_0 \subset T_k$, the row vectors $\Phi_0(x)$ and $\Phi_k(x)$ are related via $\Phi_0(x) = \Phi_k(x)U_{k0}$ where the matrix U_{k0} downsamples the vector $\Phi_k(x)$ by deleting entries that correspond to knots in T_k/T_0. Now, the following theorem holds.

Refinement theorem. Given a biharmonic B-spline defined on T_0 of the form $\mathbb{B}_0(x)p_0$, let p_k be the solution to the equations

$$N_k p_k = U_{k0} N_0 p_0, \tag{3.14}$$

$$a_k p_k = a_0 p_0. \tag{3.15}$$

Then $p_0(x) = \mathbb{B}_k(x)p_k$.

*Proof:*By the definition of $\mathbb{B}_k(x)$, the right hand side of this relation reduces to $(a_k + \Phi_k(x)N_k)\, p_k$. Applying both refinement equation to this expression yields $\Phi_k(x)U_{k0}N_0p_0 + a_0p_0$. Replacing $\Phi_k(x)U_{k0}$ by $\Phi_0(x)$ yields $(a_0 + \Phi_0(x)N_0)p_0 = \mathbb{B}_0(x)p_0$. QED

In the case of cubic B-splines, these matrices N_k are non-symmetric and compute the difference of third divided differences. As such, the refinement method corresponds to a finite difference scheme and is not a finite element scheme. In the case of biharmonic B-splines, these refinement equations have a very simple interpretation as a finite

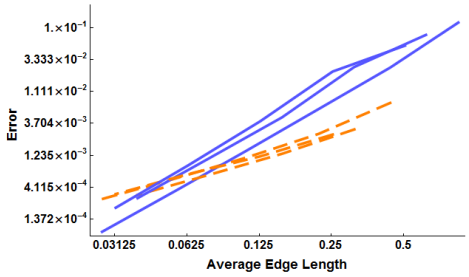


Figure 3.12 : Convergence rates for the harmonic (dashed) and biharmonic (solid) examples of figures 3.10 and 3.11.

difference scheme. Given a vector p_0, we compute the biharmonic differences $N_0 p_0$ on the knot set T_0 and then upsample these differences onto the knot set T_k via left multiplication by the matrix U_{k0}. The solution p_k to the first equation is chosen to replicate these differences on T_k. Since the preconditioners N_0 and N_k both have the constant vector in the their left null-space, the solution to this first equation is not unique. The second area equation ensures a unique solution by requiring the integrals of $\mathbb{B}_0(x)p_0$ and $\mathbb{B}_k(x)p_k$ to be equal.

Due to the fact that our refinement method is a finite difference scheme and convergence proofs for such schemes are often difficult, we can offer no formal proof of convergence at this point and instead note that the examples shown in Figures 3.10 and 3.11 were computed using this refinement scheme. We triangulated the coarse knot sets and repeatedly inserted new knots via a spherically projected variation of Loop subdivision. Figure 3.12 shows a plot of the difference between these discrete approximations at various stages during the subdivision process and the exact analytic form of the splines (as defined by the spherical Green's functions). (Of course, the harmonic scheme may diverge at the coarse knots to form poles as expected).

Figure 3.13 shows several examples of biharmonic B-spline functions on a torus. In this example, the coarse knot set has approximately 120 vertices while the fine knot set has approximately ten thousand vertices. Note that two examples on the left are

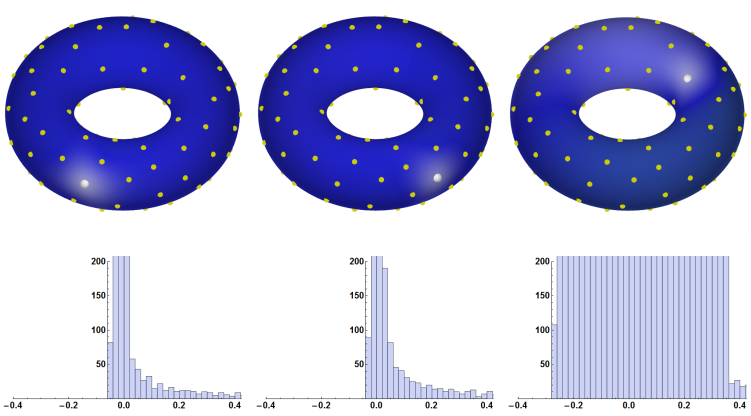


Figure 3.13 : Biharmonic B-spline functions on the torus. Plotted as temperature maps (top) and their histograms (bottom).

smooth and localized. On the other hand, the third example of the right while smooth is not as strongly localized (as shown in its histogram). In this case, we believe that this poor localization is due to the two-ring of the basis function's central vertex wrapping all the way around the inner loop of the torus. As a result, the local parametrization used in the fitting process is poor and leads to a mask that is a poor discretization of the boundary integral. (We are actively considering alternate methods for constructing a discrete mask that provides a better approximation in such cases).

3.4.2 Local editing via progressive mesh refinement

Our refinement scheme enables local manipulation of biharmonic B-splines. Given a function $p(x)$ defined on T_0, we wish to modify the shape of this function in the neighborhood of a knot t_*. If none of the knots in T_0 lie near t_*, we form a new knot set $T_1 = T_0 \cup \{t_*\}$ and compute the new coefficients p_1 that reproduce $p(x)$ on T_1 via equations 3.14 and 3.15. This new representation for $p(x)$ can then be modified in the area of t_* by varying the value of its associated coefficient in p_1 as seen in Figure

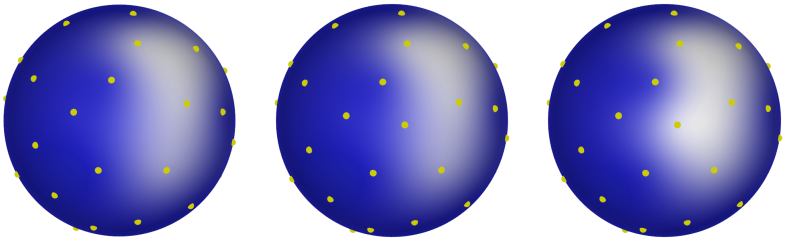


Figure 3.14 : Progressive refinement for biharmonic B-splines. Left to right; biharmonic test function (plotted as temperature map), test function after insertion of a new knot (note test function is unchanged), modification of test function at new knot (note localization of change).

3.14.

One simple strategy for performing this operation in the discrete domain is to fix a dense knot set T_k on $\mathcal{M}$ beforehand and then restrict the choice of newly inserted knots t_* to lie in T_k. In this framework, all functions defined on $\mathcal{M}$ are represented as piecewise linear functions on T_k. In fact, the underlying curved domain $\mathcal{M}$ and the piecewise linear mesh defined by the T_k may be substituted for $\mathcal{M}$ in all computations. Figure 3.14 shows an example of progressive refinement and local editing of a test function on a sphere.

3.4.3 Implementation details, timings and weaknesses

We have implemented an interactive version of the discrete refinement schemes described above for both harmonic and biharmonic B-splines using a combination of *C++* and *Mathematica*. The input to our scheme is a dense piecewise linear mesh approximating some smooth compact domain $\mathcal{M}$. The vertices of this mesh serve as the densest knot set T_k associated with this domain. If the triangles for the mesh do not form an intrinsic Delaunay triangulation, we use the edge flipping scheme of [25] to convert the mesh into such a triangulation.

Given a coarse knot set T_0 on $\mathcal{M}$, we construct the Voronoi diagrams of these knots on $\mathcal{M}$ in a manner similar to that of [51]. In particular, we compute a piecewise linear approximation $q_i(x)$ of the square of the geodesic distance from the knot t_i to a point $x \in \mathcal{M}$ with the property that $q_i(t_j)$ is the square of the exact geodesic distance from t_i to t_j on $\mathcal{M}$ (as computed in [76]). Now, an approximation to the Voronoi tile $\mathcal{V}_j$ is the set of points x such that $q_j(x) \geq q_i(x)$ for all $j \neq i$. These tiles and the lengths of their corresponding Voronoi edges can easily be computed using a standard contouring method for functions on piecewise linear meshes. This portion of the computation is done in *C++*.

Given the Delaunay triangulation, we next compute the preconditioner N. In particular, the discrete bi-Laplacian for an edge e_{ij} requires a local parametrization s_k for the knots in $t_k \in r_{ij}$ and a parametric surface $\mu(s)$ that fits these knots. We have experimented with several local parametrization methods and found that the all-pairs distance method of [12] provides the best localization in the resulting preconditioner N. Given that we have already computed these distances on $\mathcal{M}$ during the construction of the Voronoi diagram, this choice requires no extra distance computation.

Finally, the coefficients μ_α of the fitting surface $\mu(s)$ can be computed from the pseudo-inverse of the Vandermonde matrix M via $(m_\alpha) = M^+ (s_k)$ where (m_a) and (s_k) are column vectors of these coefficients. This portion of the computation is done in *Mathematica.*

Given the preconditioners N_0 and N_k for the coarse and fine knot sets, we compute dense approximations for each basis function on the coarse knot set T_0 in terms of the fine knot set T_k using equations 3.14 and 3.15. To save space and accelerate this computation in *Mathematica,* we represent these preconditioners as sparse matrices and allow *Mathematica* to choose an appropriate sparser solver. In practice, *Mathematica* solves these sparse system of linear equations in a fraction of a second for fine meshes with ten of thousands of knots. Finally, these discrete approximations to the

examples (# vertices)	all-pairs distance	fine precon.	coarse precon. (200)
sphere (2562)	276.01	105.82	1.94
sphere (10242)	9395.51	2300.05	1.92
torus (3072)	438.83	150.35	2.16
torus (12288)	16154.13	3557.93	1.88

Table 3.1 : Timings (in seconds) for main steps of our construction on various examples.

basis functions are passed back to *C++* where linear combination of these discrete approximations are interactively visualized in *OpenGL* as colors applied to the knot T_k on the fine piecewise linear mesh defining $\mathcal{M}$.

Table 3.1 shows the times for the three main steps of this process; (A) computation of all pairs distance on $\mathcal{M}$ for knots in T_k, (B) construction of the fine preconditioner N_k, and (C) construction of the coarse preconditioner N_0 and solution of equations 3.14 and 3.15. In case where T_k contains tens of thousands of knots, steps A and B are time-consuming and performed once as a pre-computation. The remaining step, comprising the heart of the editing process, can be performed efficiently (even when computed using *Mathematica*), allowing for interactive manipulating and editing of the resulting biharmonic B-splines.

Weakness The ideas described in this chapter are clearly in a partially-developed state and we can identify some issues worthy of more work.

- In the planar case, the shape properties of the biharmonic basis functions deserve much deeper examination. Although the functions are guaranteed to be smooth and localized, we have observed that the shape quality of the planar basis functions depends on the local knot geometry to some extent. Given that

our construction for the planar cubic edge mask in section 3.2 normally has several extra degrees of freedom, we believe that it should be possible to optimize the assignment of these degrees of freedom to improve the shape quality of the resulting basis functions.

- In the planar case, we did not address the issue of finite knot sets and boundaries. In practice, finite knot sets require customizing the preconditioner N at the boundary of the Delaunay triangulation so as to satisfy the structural requirements of Green's theorem. We believe that this generalization is possible and will address this problem in future work.

- In the curved case, our construction of the cubic edge mask in section 3.3 involves two components that we believe can be improved: the all-pairs geodesic distance parametrization and the approximation of the derivative of the Laplace-Beltrami operator from a locally-fitted cubic surface. The all-pairs parametrization method produces inferior parameterizations in areas where ratio of the surface curvature versus the knot density is large. In such case, we have noted biharmonic basis functions are poorly localized (such as in Figure 3.13) due to the failure of our fitting approach to capture the local shape of functions on the surface. Ideally, we seek a construction for the discrete bi-Laplacian that has the fundamental simplicity of the *cotangent* formula for the discrete Laplacian while providing provable bounds on the accuracy of the discrete approximation to the associated boundary integral.

- Finally, the use of the all-pairs distances in constructing the Voronoi tiles for T_0 is a limiting factor in our implementation and was used only as matter of convenience since we had access to the *C++* code for this method. Given the current work on computing Voronoi tiles on meshes, we believe that it should be possible to build an interactive method that avoids the need for all-pairs distances.

3.5 Discussion and future work

In this chapter, we focused mainly on the role of the discrete bi-Laplacian in defining biharmonic B-splines and considered only one application of these splines, progressive editing. The discrete bi-Laplacian and its associated preconditioner N are also an interesting candidate for discrete signal processing. Note that N is asymmetric in the case of irregularly spaced knots (just as the cubic divided differences are asymmetric). Asymmetry does not automatically preclude the use of N in discrete signal processing since its associated eigenvectors may be still be orthogonal with respect to a weighted inner product. Another possibility would be to explore the use of finite difference energy methods [60] in constructing a symmetric preconditioner N. The finite difference matrices could be designed to have the appropriate precision while the variational construction would make a proof of convergence more feasible.

Our progressive refinement scheme offers the possibility of fully irregular multi-resolution approximation of functions on planar and curved domains. One advantage of an irregular approach versus the standard regular approach is that the knot positions used in the approximation can be adjusted to adapt to the shape of the function. Figure 3.1 shows a hand-generated example of a possible method. Given a dense function defined on the sphere, we hand placed 150 knots and computed the best fitting biharmonic B-spline to this dense function. We exploited the fact that biharmonic B-splines are localized over their Voronoi tiles in positioning these knots and plan to automate this process in the near future. An alternative approach to this problem would be to start with the dense knot set/data and apply an iterative knot removal scheme similar to those of univariate B-splines. In cases like the sphere where the manifold is known, similar knot removal scheme for biharmonic B-splines should be relatively simple and should yield excellent results.

Appendix

Proof of partition of unity

Consider the effect of dilating the knot set T and the domains Ω_k (associated with the ω_k) by a factor of 2^{-k}. Under this transformation, each knot t_j is replaced by $\frac{1}{2^k}t_j$ and the dilated domains $2^{-k}\Omega_k$ form increasingly more accurate approximation to the unit disk centered at the origin. For each term of the boundary sum above, the dilation introduces several extra factors of 2^{-k} into Eq. 3.8

$$\sum_{v_{ij}\in\partial\Omega_k} \left|2^{-k}v_{ij}\right| \frac{\phi_0(2^{-k}t_i) - \phi_0(2^{-k}t_j)}{|2^{-k}e_{ij}|}$$

This expression is equivalent to the right hand side of Eq. 3.8 since the expression $\phi_0(2^{-k}t_i) - \phi_0(2^{-k}t_j)$ reduces to $\phi_0(t_i) - \phi_0(t_j)$ and the factors of 2^{-k} cancel. Given that the edge e_{ij} has bounded length, the expression $(\phi_0(2^{-k}t_i) - \phi_0(2^{-k}t_j))/\left|2^{-k}e_{ij}\right|$ converges quadratically to $\frac{\partial(\phi_0(x))}{\partial(\nu_{ij})}$ where ν_{ij} is the unit vector in direction e_{ij} and $x \in 2^{-k}v_{ij}$. Thus, the boundary sum of Eq. 3.8 converges linearly to the boundary integral of the form

$$\sum_{v_{ij}\in\partial\Omega_k} \mathbb{Z}_{x\in 2^{-k}v_{ij}} \frac{\partial(\phi_0(x))}{\partial(\nu_{ij})} ds.$$

Now, this boundary integral reduces an equivalent area integral over $2^{-k}\Omega_k$ which by Green's theorem is exactly one. QED.

Computation with the Laplace-Beltrami operator

Let $\mu(s)$ be parametrization of a manifold $\mathcal{M}$ and let $p(s)$ be a function on $\mathcal{M}$. The Laplace-Beltrami operator Δ_B of $p(s)$ with respect to the parametrization $\mu(s)$ has the form [17]

$$\frac{\mathrm{div}(\sqrt{\mathrm{Det}(\mathrm{g}(\mu))}\mathrm{g}(\mu)^{-1}\nabla(p))}{\sqrt{\mathrm{Det}(\mathrm{g}(\mu))}}$$

where div(f) (a row vector) is the divergence of a vector-valued function $f(s)$ and g(μ) (a matrix) is the first fundamental form of the parametrization $\mu(s)$.

Given a multi-index α, our task is to compute a directional derivative of the Laplace-Beltrami operator, applied to a monomial s^α. In particular, we wish to compute the expression $\frac{\partial(\Delta_B s^\alpha)}{\partial \nu}$ with respect to the parametrization $\mu(s)$ where ν is unit vector in the s-plane. Recalling that the coefficients μ_β of the cubic surfaces $\mu(s)$ are triples (each entry corresponding to a coordinate of the knot). Applying this definition to $\frac{\partial(\Delta_B s^\alpha)}{\partial \nu}$ yields a rational function whose terms consist of products of dot products between various coefficient vectors μ_β. These rational functions can be pre-computed symbolically once in *Mathematica* and then evaluated efficiently using the coefficients μ_β computed during the surface fitting process.

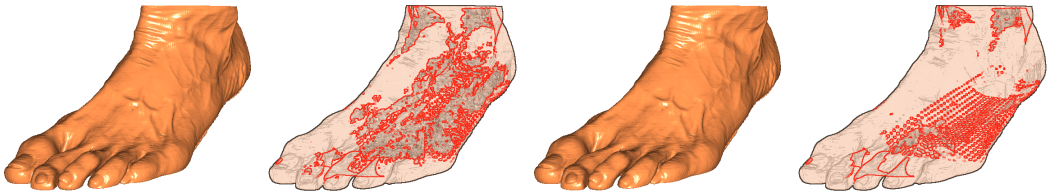
Chapter 4

View-independent Contour Culling of 3D Density Maps for Far-field Viewing of Iso-surfaces

4.1 Motivation

Iso-surfaces are commonly used for visualizing 3D density maps, such as MRI and CT scans in bio-medical applications. With the increasing complexity of today's imaging data, contouring a density map easily yields iso-surfaces with high polygon counts that are costly to produce, store and render. Often times, however, the number of elements contained in the iso-surface does not proportionally correspond to visual complexity perceived by the viewer. Consider the iso-surface in Figure 4.1 (a) from the CT scan of a human foot. There are many interior surface pieces, which are highlighted in Figure 4.1 (b). These interior parts account for close to half of the total triangles in the iso-surface, and yet they are not visible if one views the foot from the outside.

One method for handling this issue is to cull invisible components of the iso-surfaces at rendering time using existing *visibility culling* methods [13]. While this approach is feasible, the extraction and storage of the original, un-culled iso-surface is still required. A better solution would be to perform culling at the contouring stage, thus avoiding the generation of invisible surface components in the first place. We call this second approach *contour culling*. While there have been a number such methods proposed, these methods are all *view-dependent* (see a brief review in the next section). This means that culling has to be performed whenever the user changes the viewing angle. As a result, a significant overhead can be added to the run-time



(a) (b) (c) (d)

Figure 4.1 : (a,c): Iso-surfaces of a CT foot scan at the same iso-value generated by Marching Cubes without and with culling, containing 1020570 and 592456 triangles respectively. (b,d): Transparent rendering of the polygons in (a,c) (see details in Section 4.5), showing internal structures in the original iso-surface that have been largely culled away using our technique.

rendering pipeline.

4.1.1 Contribution

In this chapter, we explore an alternative, *view-independent* contour culling approach that adds little computational overhead at run-time. We consider a specific but common scenario of iso-surfaces visualization, which we call *far-field viewing*. In this scenario, the user views the iso-surfaces from a distance (e.g., the view taken in Figure 4.1 (a)) and varies the iso-value associated with the iso-surface as well as the viewpoint and viewing direction to gain a sense of the general 3D structure of the density map. At each iso-value, our method culls parts of the iso-surface invisible to *any* viewpoints located outside the volume (e.g., highlighted parts in Figure 4.1 (b)). Once the iso-surface is generated, it can be viewed from different directions with no more computations needed other than rendering.

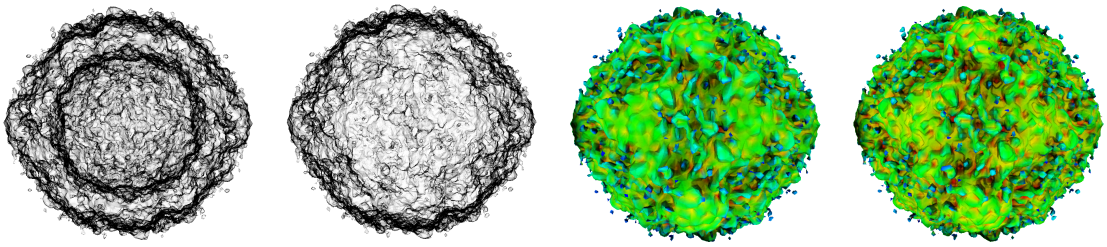
There has been active research into visibility culling methods that cull based on views

from a region rather than from a single point (see a brief survey in [64]). However, these methods cannot be directly applied to iso-surface visualization as they usually require considerable pre-computation that is specific to the surface to be culled. When the user changes the iso-value, such pre-computation needs to be performed again.

In contrast, our method performs a single pre-processing step for iso-surfaces at *all* iso-values. The key observation is that values in a density map (especially bio-medical images) typically drop off at the boundary of the map. During far-field viewing of such a map, a point x in the map that is completely invisible when viewing from outside the map at some iso-value c will also stay invisible for iso-values lower than c. We call the minimal iso-value that x is visible to some outside views the *contour visibility function* (CVF), or $g_f(x)$ (detailed in Section 2). With a pre-computed CVF, the visible parts of the iso-surface at any iso-value c can be easily extracted by contouring only in parts of the map where $g_f(x) \leq c$.

Our main contribution is a simple dynamic-programming algorithm for computing a discrete approximation of CVF for tri-linearly interpolated 3D density functions (see Section 3). We show that this piece-wise constant approximation can be easily utilized by a tri-linear contouring algorithm or typical polygonal contouring algorithms (such as Marching Cubes [54]) to perform culling, and the result is guaranteed to be *conservative*: the culled portions of the iso-surface are not to the viewer at any view angle for far-field viewing (see Section 4).

An example result of our method is shown in Figure 4.1 (c). Note that the visible parts of the iso-surface are well preserved, whereas a significant amount of interior pieces are removed (see (d)). When tested on several real-world bio-medical data (see Section 5), we observed that our view-independent culling approach performs well for density maps that contain large inner iso-surface pieces, sometimes achieving up to 80% surface reduction.



(a) (b) (c) (d)

Figure 4.2 : This is a poliovirus with an inner surface not visible from a far-field view. All meshes are generated from a 201^3 volume and simplified to 100K triangles. The mesh generated from the original density data contains an inner surface (a,c), and the one generated from our contour-culled density shows that the inner surface has been removed (b,d). Images (c) and (d) show the meshes colored by curvature. By comparing the variation in curvature, we see that the contour-culled version retains more surface details than the mesh from the original volume.

4.1.2 Application

The reduced run-time overhead of our view-independent culling approach applies naturally to online and mobile platforms with limited access to computational and rendering resources. As a concrete example, we incorporated the culling algorithm into an online visualization applet for 3D density maps of macromolecular structures (such as viruses) as part of the Electron Microscopy Databank (EMDB) [46]. The iso-surfaces of such data often contain large nested, interior portions, and the biologists most often take views from outside of the volume. Hence our culling method is particularly suited. In this application, both contouring (using Marching Cubes) and culling are performed on the server side, and the resulting surface is transformed to the client-side applet for viewing. Due to bandwidth limits, the iso-surface needs to be simplified before transmission [32]. With culling turned on, simplification becomes more efficient (since there are fewer triangles to start with), and the simplified

result better preserves the details on the visible parts of the iso-surface. This is demonstrated in Figure 4.2. Note that since our culling is view-independent, no extra computation is needed on the client-side applet as the user inspects the surface from different views.

4.1.3 Related work

Visibility culling is a well-studied problem in computer graphics. Previous work on visibility culling has focused on the problem of culling the invisible portions of a polygonal mesh with respect to a view direction. Classic techniques that try to address the visibility culling problem include Z-buffer, back-face culling, view-frustum culling, and visibility space partitions (see the survey [13]). Our approach differs from traditional approach in that we focus on avoiding the generation of unnecessary polygons when extracting level-sets from a density map. Most of the previous methods are compatible with our method since they can be applied after a polygonal mesh has been extracted from the density map.

Several algorithms for culling iso-surfaces based on visibility have been proposed in the past. Given an iso-value, these algorithms generate only part of the iso-surfaces visible to the viewer. Livnat and Hansen [52] perform a front-to-back sweep of an octree representation of the density volume to quickly identify regions of the space occluded by the iso-surface. In a similar approach, Gao and Shen [30] improve the culling performance on multiple processors using an image space culling algorithm and a load-balanced workflow. Pesco et al. [65] uses an implicit culling scheme based on the scalar values at the voxels instead of actual triangles on the iso-surface. Recently, Gregorski et al. proposes a culling algorithm for iso-surfaces extracted from hierarchical tetrahedral meshes that exploits frame-to-frame coherence between consecutive views [35].

While theses methods can dramatically reduce the number of polygons that need

to be rendered, their view-dependence requires culling and surface extraction to be performed whenever the view is changed. In addition, culling in these methods can add significant overhead (in both implementation complexity and running time) to the contouring algorithm and is specific to the iso-value used to generate the iso-surface. In contrast, our approach does not require re-generation of iso-surfaces as the view changes, and our method adds little overhead to the contouring algorithm regardless of the iso-value used.

Our work is closely related to methods that compute occlusion maps to accelerate rendering. One of the earliest works is by Zhang et al., who propose computing a hierarchy of occlusion maps to accelerate the rendering of scenes with large number of primitives [86]. Our work differs from Zhang et al's work in that our method targets density maps. In particular, our visibility function can be viewed as an occlusion map, but it performs culling for all iso-values. Li et al. extended the idea of occlusion maps into the domain of volume rendering. In their method, the occlusion map is updated at every frame (i.e. view dependent). In contrast, we compute a single, conservative occlusion map for all-directions [47].

Also related to contour culling is the class of methods that cull occluded voxels for volume rendering [29, 31]. Like iso-surface culling, most of these methods are view-dependent. A notable exception, and one that is most related to our work, is the works by Mroz and colleagues [59, 58] that perform culling for Maximum Intensity Projection (MIP), a rendering method that displays the maximum intensity along each ray through the volume. Similar to us, the authors observe that a (possibly large) fraction of the voxels do not contribute to the MIP image at *any* viewing angle. Hence they identify such voxels in a pre-processing stage and preclude them in the rendering stage. The identification is done either by recursive ray search at each voxel [59], or by a front-moving algorithm [58] that is similar in spirit to the dynamic programming algorithm proposed in this chapter.

There are a number of key differences between our work and that of Mroz and colleagues. First, the two works are designed for different visualization modes (MIP versus iso-surfaces), hence the culling criteria is different. Second, while MIP culling creates a binary mask (kept or removed) for the voxels, our approach builds a floating-point visibility function over the volume that provides the culling information at all iso-values. Third, comparing to front-moving in [58], our dynamic programming algorithm offers the additional control over the aggressiveness of surface reduction while maintaining a conservative culling.

4.2 The contour visibility function

We proceed to provide a mathematical basis for our technique. We will first formulate the contour visibility function (CVF), which gives the exact range of level sets at which a point in space is visible when viewed at infinity. We will then consider approximations of such functions with conservative guarantees, which allow efficient computations (to be discussed in Section 4.3).

4.2.1 The definition

Defining the CVF of a given density function $f(x)$ is fairly straightforward. Assuming again that the values of $f(x)$ lie in the range $[0, 1]$ and that tend towards zero as $x \to \infty$. Let r_x be a ray from x to infinity and f_{r_x} be the maximum of f taken at all points (including x) along the ray r_x (i.e. $\max_{z \in r_x} f(z)$). We define the CVF, denoted as $g_f(x)$, to be the minimum of f_{r_x} over all possible rays r_x.

We now claim that given some iso-surface $f(x) = c$ where $0 < c < 1$, a point x (which may or may not be on the iso-surface) lies in the visible space when viewed from infinity in some direction if and only if $g_f(x) \leq c$. To prove this, note that a ray intersects with the iso-surface if and only if there are some values along the ray

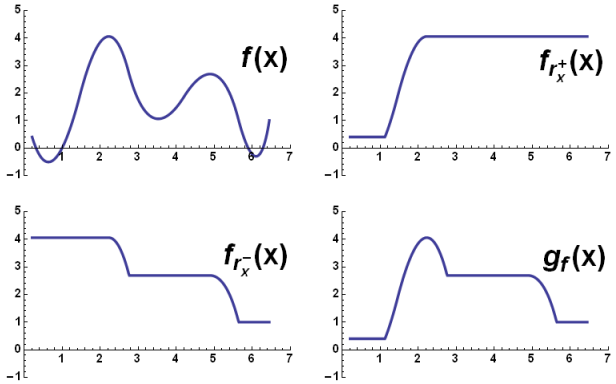


Figure 4.3 : The contour visibility function in 1D: $f(x)$ is the original function, $f_{r_x^+}, f_{r_x^-}$ are maximum values along the two rays respectively from x to $+\infty$ and $-\infty$, and $g_f(x)$ is the minimum of $f_{r_x^+}$ and $f_{r_x^-}$.

above c and some below c (assuming the intersection is in a general position and that the iso-surface avoids the singularity of f). Since f tends to zero at infinity, a ray r_x intersects with the iso-surface if and only if $f_{r_x} > c$. As a result, x is un-blocked by the iso-surface in some direction if and only if $f_{r_x} \leq c$ for some r_x, and likewise $g_f(x) \leq c$.

Figure 4.3 illustrates the CVF of a 1D function (shown in top-left). In 1D, there are only two possible rays associated with each point x that go toward either the positive or negative infinity, which we denote respectively as r_x^+ and r_x^-. The ray maxima $f_{r_x^+}, f_{r_x^-}$ are plotted respectively in top-right and bottom-left for each x, and the CVF $g_f(x)$, the minimum of the two ray maxima, is plotted in bottom-right. Observe that, by our definition, $f(x) \leq g_f(x)$.

4.2.2 Conservative approximations

While computing CVF is relatively straightforward in 1D, it is much more difficult in 2D and 3D . Even in the case where $f(x)$ has a simple structure, such as being piece-

wise constant, the function $g_f(x)$ is likely to have a much more complex structure that is challenging to compute both accurately and efficiently.

From the computational perspective, we are mostly interested in approximations of the exact CVF. In particular, we are interested in approximations $g(x)$ that are *conservative*: given any iso-surface $f(x) = c$, a point x such that $g(x) > c$ should never be visible in any direction when viewed from infinity. In this way, culling away all points x on the iso-surface where $g(x) > c$ will not affect the visible part of the surface in any views (even though some hidden, interior parts may still remain).

It is easy to show that an approximation $g(x)$ is conservative if $0 < g(x) \leq g_f(x)$ for any x. The closer $g(x)$ approaches $g_f(x)$, the tighter the approximation, and larger portions of the iso-surfaces could be culled using $g(x)$. Note that $f(x)$ itself is a conservative approximation, as $f(x) \leq g_f(x)$. Intuitively, $f(x) > c$ meaning that x lies in the "inside" of the iso-surface $f(x) = c$, hence invisible to the outside.

4.3 Computing the approximated CVF

In many contouring applications, the density map is provided as scalar values associated with points on a 3D integer grid. These values are then interpolated to form $f(x)$, typically using tri-linear interpolation. Our goal in this section is to develop an efficient algorithm that computes a conservative approximation of the CVF in a tri-linearly interpolated density function.

4.3.1 Motivation

The algorithm is motivated by the piece-wise nature of tri-linear interpolation.The interpolated function $f(x)$ is made up of smooth pieces within each grid "cell" formed by eight grid points. Likewise, tri-linear contouring algorithms are typically performed in a cell-by-cell manner, producing one piece within a cell. If we know which cells

are invisible, we can easily modify the contouring algorithm to skip the invisible cells. To make such decisions quickly and accurately, all we need is a conservative approximation to $g_f(x)$ that assumes a *constant* value in each cell. If this value is greater than the current iso-value, the conservativeness guarantees that the entire cell is never visible in any view directions.

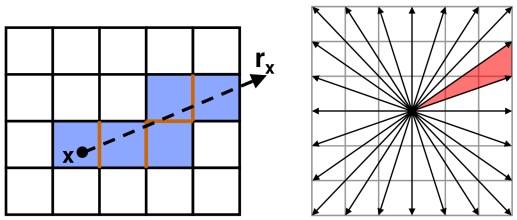
For ease of explanation, we will start by describing the algorithm for computing this piece-wise constant approximation of CVF in 2D. We will then show how the implementation can be extended to work on 3D volumes.

4.3.2 Algorithm in 2D

Consider the problem of approximating the CVF for a 2D density function $f(x)$ over an integer grid with $n \times n$ squares such that the function gives a constant value $g_{i,j}$ for each grid square $p_{i,j}$. To be a conservative approximation, we ask $g_{i,j} \leq g_f(x)$ for any $x \in p_{i,j}$ where $g_f(x)$ is the actual CVF.

We start by examining the discrete path of grid squares, denoted as $r_{i,j}$, that is intersected by the ray r_x from some point $x \in p_{i,j}$ to infinity (see the shaded squares in Figure 4.4 (a)). Observe that consecutive squares in the path share common edges (highlighted in the picture) that are intersected by the ray. Now, consider the minimum of f along each of these common edges, and let $f_{r_{i,j}}$ be the maximum of such minima over all edges along the path $r_{i,j}$. We first point out that $f_{r_{i,j}}$ is a lower-bound of f_{r_x}, which is the maximum of all values of f along the continuous ray r_x. With this observation, we can see that a conservative choice of $g_{i,j}$ would be the minimum of $f_{r_{i,j}}$ over all possible discrete paths $r_{i,j}$ whose defining rays come out of some point $x \in p_{i,j}$.

In the parlance of digital geometry, the path of grid squares intersected by a straight ray is known as a *digital straight line* [44]. To compute $g_{i,j}$ as described above, we



(a) (b)

Figure 4.4 : (a) A ray r_x and the digital straight line it defines (shaded) which consists of edge-adjacent squares. (b) The partition of the view angles into view cones. The highlighted triangle is a single view cone.

could just enumerate all digital straight lines starting from $p_{i,j}$. Unfortunately, as shown in [44] and [1] there are $O(n^3)$ digital straight lines that pass through a given grid square, so that approach does not seem to be a feasible method for computing $g_{i,j}$ for all n^2 grid squares. In 3D (our ultimate goal), the situation appears to be even worse. Instead, our approach will compute a lower bound of $g_{i,j}$ as described above, by taking the minimum of $f_{r_{i,j}}$ over a larger set of edge-adjacent square paths $r_{i,j}$ coming out of $p_{i,j}$ that includes all the digital straight lines. Unlike digital straight lines, this expanded set of paths can be enumerated much more efficiently using local operations.

The algorithm starts by partitioning all view angles into a finite number of *view cones*. For each view cone, we compute at each grid square $p_{i,j}$ a lower bound of $f_{r_{i,j}}$ for all digital straight lines $r_{i,j}$ whose defining rays lie in the view cone. This is done using an efficient two-part dynamic programming algorithm. Performing the algorithm for each view cone yields one value at each grid square $p_{i,j}$, and finally $g_{i,j}$ is taken as the minimum of such values computed at $p_{i,j}$ over all view cones. The algorithm is detailed below and summarized in the pseudo-code in Program 1.

Partitioning the view angles

To partition all view angles into a set of view cones, consider a $2m \times 2m$ integer grid centered at the origin as shown in Figure 4.4b (where $m = 3$). Forming vectors from the origin to the boundary grid points yields a set of $8m$ vectors. Each pair of adjacent vectors defines a view cone for those ray whose slopes lie in the given cone. For example, the shaded cone in Figure 4.5 is bounded by the vectors $(3, 1)$ and $(3, 2)$.

Approximating CVF within each view cone

Without loss of generality, we now focus on the computation for a single view cone. For the sake of simplicity, we consider the view cone that lies in the lower right half of the first quadrant and is bounded by the rays (m, l) and $(m, l + 1)$ where $0 \leq l < m$. (The remaining cases can be converted to this case via the appropriate horizontal, vertical or diagonal reflection of f.)

Our goal is to compute at each grid square (i.e. $p_{0,0}$) a lower bound of $f_{r_{0,0}}$, noted as $g_{0,0}$, for all digital straight lines $r_{0,0}$ defined by rays in the view cone. Our key observation here is any segment of a digital straight line in the view cone is also a digital straight line in the same view cone. We thus compute the desired lower bound $g_{0,0}$ in two stages. In the first stage, we consider all digital straight lines $r_{0,0}$ with horizontal span m in the view cone. Note that such lines must end either at the grid square $p_{m,l}$ or $p_{m,l+1}$. We obtain the lower bound of $f_{r_{0,0}}$ over all $r_{0,0}$ ending at $p_{m,l}$ and $p_{m,l+1}$ respectively as $h_{m,l}$ and $h_{m,l+1}$ (elaboration to follow). In the second stage, we obtain the final lower bound $g_{0,0}$ for all digital straight lines in the view cone (with horizontal spans no smaller than m) using a simple dynamic programming pass. Assuming $g_{(m,l)}$ and $g_{(m,l+1)}$ have been computed correctly, we compute

$$g_{(0,0)} = \min\left(\max\left(h_{m,l}, g_{m,l}\right), \max\left(h_{m,l+1}, g_{m,l+1}\right)\right) \tag{4.1}$$

To obtain the lower bounds $h_{m,l}, h_{m,l+1}$, one could enumerate the digital straight lines

Partition the view cone into $8m$ cases

/* m is a small non-negative integer*/

For each view cone

Flip f vertically, horizontally or diagonally
such that the view cone is bounded by the
vectors (m, l) and $(m, l+1)$ where $0 \leq l < m$

Let S be set of squares whose intersection with the convex hull
of $p_{0,0}$, $p_{m,l}$ and $p_{m,l+1}$ is non-empty

Let $g_{i,j} = 0$ for $n \leq i < n+m$ and $n \leq j < n+l+1$

/* g is an $n \times n$ density grid */

For $i = n-1$ to 0 by -1 and $j = n-1$ to 0 by -1

Let $h_{0,0} = 0$

For $c = 0$ to m by 1 and $d = 0$ to $l+1$ by 1

If $p_{c,d} \in S$

Let $h_{c,d} = \min\left(\max\left(f_{(i+c)^-,j+d}, h_{c-1,d}\right), \max\left(f_{i+c,(j+d)^-}, h_{c,d-1}\right)\right)$

Let $g_{i,j} = \min\left(\max\left(h_{m,l}, g_{i+m,j+l}\right), \max\left(h_{m,l+1}, g_{i+m,j+l+1}\right)\right)$

Perform the inverse of the flips applied f to g

Let the final g be the minimum of the g computed for each view cone

Program 1: Pseudo-code for approximating the CVF for a 2D piece-wise density function f.

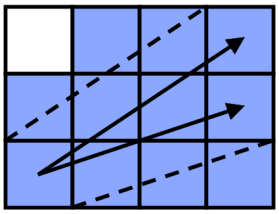


Figure 4.5 : Grid squares that intersect with the convex hull of a view cone (marked by the dotted line).

starting at $p_{0,0}$ and ending at either $p_{m,l}$ or $p_{m,l+1}$. Since m is small, such enumeration wouldn't be so costly. Alternatively, one can use a dynamic programming scheme similar to the above description to compute a more conservative bound, which is implemented here. Specifically, let us consider the set of all edge-adjacent square paths $r_{0,0}$ starting from grid square $p_{0,0}$, moving only rightward or upward, ending at either $p_{m,l}$ or $p_{m,l+1}$, and visiting only those grid squares intersecting with the convex hull of squares $p_{0,0}, p_{m,l}, p_{m,l+1}$. An example of the allowed squares for $m = 3, l = 2$ is shown shaded in Figure 4.5). Note that these paths include all digital straight lines with slope bounded by the view cone. These paths can be enumerated, and the minimum of $f_{r_{0,0}}$ over these paths can be updated in a single dynamic programming pass. Starting with $h_{0,0} = 0$, we compute $h_{c,d}$ for all allowed squares $p_{c,d}$ inductively as

$$h_{c,d} = \min\left(\max\left(f_{c^-,d}, h_{c-1,d}\right), \max\left(f_{c,d^-}, h_{c,d-1}\right)\right) \tag{4.2}$$

where $f_{c^-,d}$ denotes the minimum of f along the edge shared by squares $p_{c,d}$ and $p_{c-1,d}$.

Overall, the dynamic programming sweep in Eq. 4.1 processes the squares $p_{i,j}$ in decreasing values of i and j. To apply this recurrence near the right and upper boundaries of the grid, we pad the grid with $l + 1$ rows and m columns of zeros. At each square $p_{i,j}$, the dynamic programming pass in Eq. 4.2 is performed to supply the outer dynamic programming sweep with the necessary quantities $h_{m,l}, h_{m,l+1}$ (see the

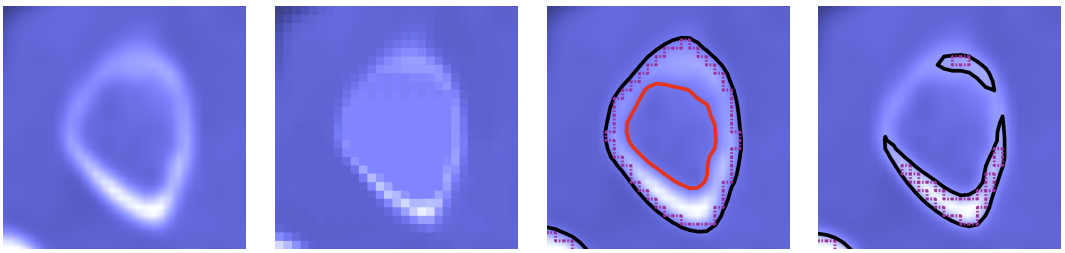
pseudo-code in Program 1).

4.3.3 Generalization to 3D

To extend the algorithm to a 3D density function $f(x)$, we will similarly compute an approximation to the CVF $g_f(x)$ that is constant within each grid cell. The value $g_{i,j,k}$ for a cell $p_{i,j,k}$ is computed as a lower bound of $f_{r_{i,j,k}}$ over all digital straight lines $r_{i,j,k}$ (consisting of face-adjacent cells intersecting some ray) passing through $p_{i,j,k}$. Here, $f_{r_{i,j,k}}$ is the maximum of the minimum of f on each grid face between successive cells on $r_{i,j,k}$. Using a similar argument as in 2D, such choice of $g_{i,j,k}$ is a conservative approximation, i.e., $g_{i,j,k} \leq g_f(x)$ for any $x \in p_{i,j,k}$.

To compute $g_{i,j,k}$, the 2D dynamic programming algorithm only needs to be slightly modified. To partition the view angles, we use a cube of size $2m \times 2m \times 2m$ centered at the origin, whose boundary is divided into $24m^2$ unit squares. Each unit square defines a view pyramid bounded by four vectors of the form (m, l, o), $(m, l+1, o)$, $(m, l, o+1)$, and $(m, l+1, o+1)$ where $0 \leq l, o < m$. The rest of the algorithm proceeds as before.

Complexity: The most time-consuming step is the inner dynamic programming pass (Eq. 4.2), which has the complexity of $O(m)$ (equalling the number of cells intersecting with the convex hull of the view pyramid, which is bounded by mk for some constant k). This pass needs to run once for each grid cell and once for each view pyramid, hence the total asymptotic complexity is $O(n^3m^3)$.The algorithm has an "embarrassingly parallel" structure, as the computation for each of the $24m^2$ view pyramids is completely independent of each other. Hence the practical performance can be easily improved by employing multiple processors (or cores).



(a) (b) (c) (d)

Figure 4.6 : 2D example of contour culling. (a): A tri-linearly interpolated density function (from the Foot data). (b): The $g_{i,j}$ computed by our algorithm for each grid square. (c,d): iso-curves of (a) at low and high iso-values, where interior curve parts invisible to the outside (red) are culled away. Purple dots outline those grid squares whose $g_{i,j}$ are greater than the iso-value, and hence where the culling takes place.

4.4 Contour culling using CVF

Our algorithm outputs a 3D visibility volume containing one value $g_{i,j,k}$ for each grid cell $p_{i,j,k}$ in the input volume. For a tri-linear contouring algorithm to take advantage of this output, simply skip cells whose g values are greater than the iso-value. The conservativeness of our computation ensures that culling does not affect the visible part of the iso-surfaces in any views. This is illustrated in 2D in Figure 4.6. Note that the interior part of the iso-curve (red curve in (d)) is culled during contouring, since the $g_{i,j}$ values of the enclosing grid squares are greater than the iso-value (such grid squares are outlined by purple dots).

The visibility volume can also be utilized by polygonal contouring algorithms that approximate the tri-linear contours and proceed in a similar cell-by-cell fashion, such as Marching Cubes. Culling in these algorithms based on $g_{i,j,k}$ is also conservative (i.e. the visible part of the polygonal surface is not affected in any views) if the polygonal contours satisfy a mild condition: a grid face whose scalar values at all four corners are above the iso-value always lies inside the polygonal iso-surface. Note that

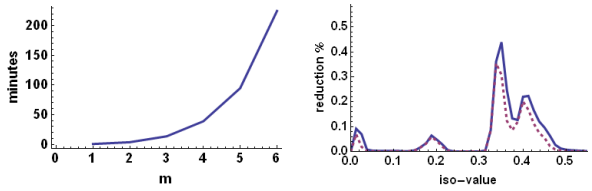
this condition holds in most algorithms like Marching Cubes.

4.5 Results

We demonstrate our algorithm on a suite of synthetic data as well as real-world biomedical images (e.g., MRI, CT, cryo-EM). To quantify the effectiveness of culling, we consider the percentage of total triangles generated by Marching Cubes [54] that are culled using our approach (as described in Section 4.4) and refer to this as the *reduction rate*.

Choice of m: This is the only parameter of our algorithm, which determines the number of view partitions. Increasing m results in a better approximation to the actual CVF because more and finer view cones (pyramids) are explored and because longer segments of discrete paths are used to approximate digital straight lines. However, larger m also significantly increases the computational cost due to the m^3 part in the algorithm complexity (see Section 4.3). In practice, we notice that higher values of m typically do not yield significantly better reduction rate. This is demonstrated in Figure 4.7, which shows the computational cost for increasing m on the Foot example (Figure 4.1) and the reduction rate at all iso-values for $m = 1$ and $m = 6$. In our examples, we used $m = 2$ which provides a good balance between the amount of reduction and running time. Figure 4.8 shows the corresponding visual impact of the m parameter.

Results: Figure 4.9 shows the graph of reduction rate for all test examples in this paper. Each curve represents a single density map. The "test128" example is synthetically created by randomly perturbing a constant density volume. The graph shows that the reduction can react sharply with respect to the iso-value, with best rates typically occurring in the middle range. In some cases, it is possible to achieve up to an 80% reduction in polygon counts.



(a) (b)

Figure 4.7 : (a): Running time of our algorithm on the Foot data as m increases. (b): Reduction rate at all iso-values for $m = 6$ (blue solid line) and $m = 1$ (purple dashed line).

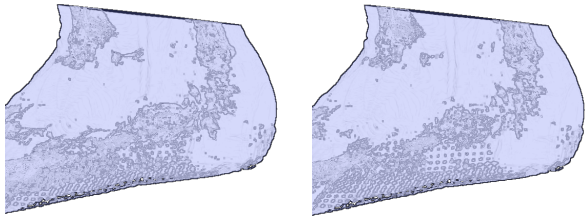


Figure 4.8 : The foot data iso-contoured for $m = 2$ (a) and $m = 6$ (b). The chosen iso-value is .351. Modified Marching Cubes generated 884135 and 778579 triangles for $m = 2$ and $m = 6$ respectively.

Table 4.1 visually shows the amount of reduction (for one iso-value) in these test examples. To visualize the hidden surface parts, we used depth-peeling and line drawing to reveal the inner layer of each mesh extracted with Marching Cubes. This rendering technique, called blueprint rendering, was described by Niehaus and Dollner [63]. We modified their original algorithm to produce red lines for the inner layers of the depth-peeled image. Generally, the inner layers reveals the inner polygons that are invisible when viewed from infinity. Table 4.2 shows culling on one example (a cryo-EM image of a virus) at different iso-values.

Performance: Our algorithm is tested on an Intel Xeon machine of 8 cores with clock-rate of 2.5Ghz. As mentioned in the complexity analysis, our algorithm is easily

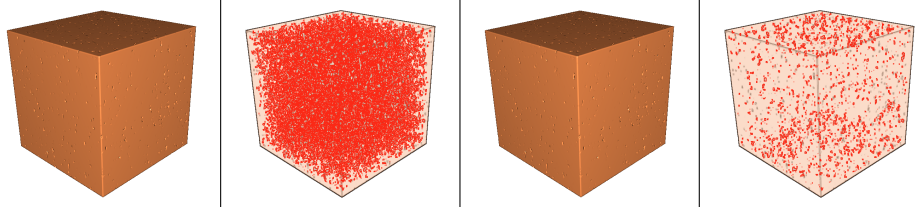
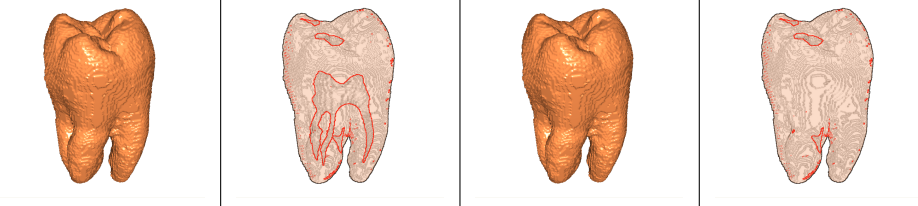
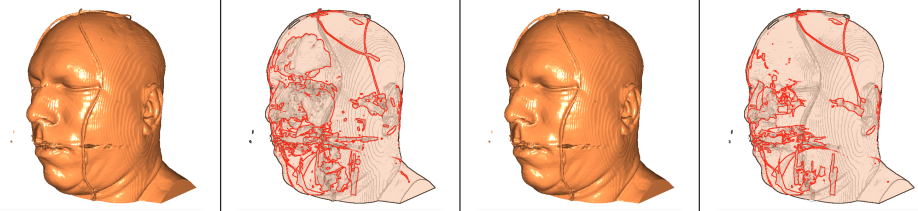
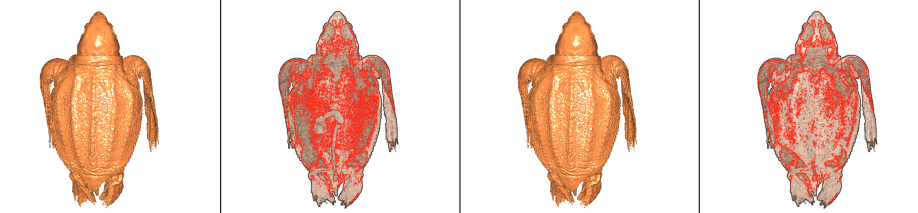
	original	original +transparent	culled	culled + transparent	reduc.
test128					
	989984 tri.		265048 tri.		73.2%
tooth					
	128908 tri.		112232 tri.		12.9%
vismale					
	703636 tri.		526792 tri.		25.1%
turtle					
	2174216 tri.		1361838 tri.		37.4%

Table 4.1 : Comparisons of iso-surfaces extracted by Marching Cubes on several data sets without culling (first column) and with culling (third column) at a single iso-value. The transparent renderings (second and fourth columns) reveal the internal portions.

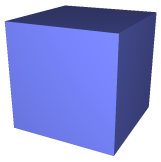
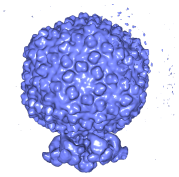
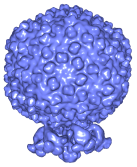
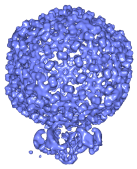
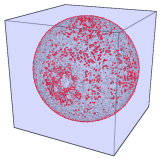
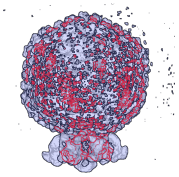
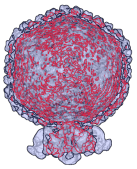
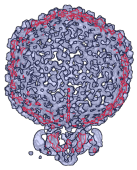
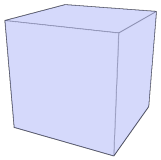
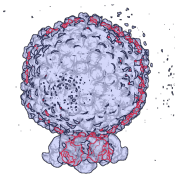
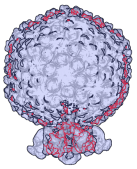
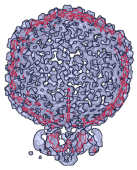
iso-value	0.44	0.50	0.55	0.74
far-field view				
without culling	 5935276 tri.	 597426 tri.	 1504356 tri.	 527112 tri.
with culling	 988428 tri.	 446166 tri.	 430450 tri.	 527112 tri.
reduc.	83.3%	25.3%	71.4%	0%

Table 4.2 : Contour culling on virus particle *e1175* (Epsilon 15) from the Electron Microscopy Data Bank (EMDB) at various iso-values, showing the iso-surface extracted by Marching Cubes (top row), internal portions before culling (second row) and after culling (last row).

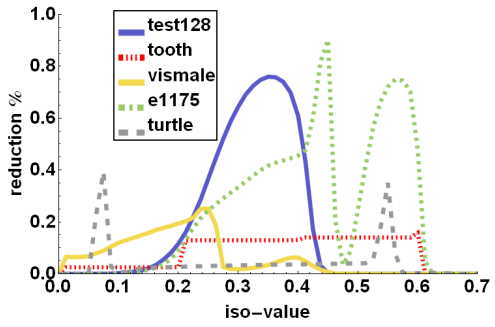


Figure 4.9 : Reduction rate of culling on several datasets at varying iso-values.

# of cores	1	2	4
time (min)	13.43	6.91	3.64

Table 4.3 : Running time on the Foot data using multiple cores (with $m = 2$).

parallelizable. Table 4.3 shows the execution time with respect to the number of cores used in the computation, exhibiting an expected linear speed-up. Table 4.4 shows the timing results for all examples, using 4 cores with our implementation. Observe that the computation on our largest dataset ($256 \times 256 \times 397$) finishes in less than 10 minutes. Note that the computation is done once for each volume, after which the only operation needed during contouring is a scalar value comparison for each cell to determine its visibility.

4.6 Discussion

We have described a novel contour culling method for far-field viewing of density maps that reduces the complexity of surfaces extracted by standard contouring techniques without altering their visual appearance. The algorithm pre-computes a visibility function (CVF) that is independent of view directions and considers all iso-values. The function can be used in conjunction with contouring to cull internal surface

dataset	size	contour culling (min)	MC (sec)	mod-MC (sec)
test128	$128 \times 128 \times 128$	.46	.229	.168
vismale	$256 \times 256 \times 128$	1.86	.092	.100
tooth	$256 \times 256 \times 161$	2.42	.172	.150
foot	$203 \times 418 \times 189$	3.64	.159	.156
e1175	$288 \times 288 \times 288$	5.29	.204	.178
turtle	$256 \times 256 \times 397$	6.10	.185	.191

Table 4.4 : The running time of our algorithm on all datasets. The timings for running regular marching cubes and our modified marching cubes are shown in the rightmost two columns. The results suggest no significant difference in running time.

components at any iso-value with little overhead added to the contouring method. The effectiveness of the approach is demonstrated on several bio-medical data, and an application for online viewing of molecular density maps is presented.

Obviously, our culling method is not a replacement for existing view-dependent methods, which work for any viewpoints and also cull more triangles from any particular view point than our method. Our method offers some unique features in the far-field viewing scenario, such as the minimal run-time computational cost. It is also simple to incorporate it into any contouring implementations (the only addition is a comparison test between values in the data and values in the visibility map). These features make our method suited for light-weight visualization tools, which are explored in this paper, and also allow potential integration of our method into existing view-dependent culling systems to offer improved culling efficiency when the viewpoint is taking from outside the volume (since fewer triangles need to be considered for culling at any particular view).

For future work, we will explore other methods for enumerating digital straight lines that provide a better coverage and lower complexity than the current dynamic programming algorithm. In this work, we were not able to provide an error bound on our approximation to the ideal CVF, and proving such bound might be another possibility for future research. Additionally, we would like to further explore the trade-off space of the accuracy of rendering versus the amount of culling. While we only consider conservative culling in this paper, in most visualization applications the conservative constraint can be relaxed to allow more aggressive culling with a small sacrifice in visual accuracy.

Chapter 5

A Dual Method for Constructing Multi-Material Solids from Ray-Reps

5.1 Motivation

The problem of reconstructing a geometric shape from some partial representation is a classical one. Techniques for reconstructing 3D shape from grid-based samples such as contouring [54, 42] are well known. Techniques for reconstructing 3D shape from a sequence of 2D cross sections such as [6, 2] and [50] are also well studied. One reconstruction problem that is not as well studied is that of reconstructing a 3D shape from a sequence of intervals attached to rays in a 2D grid of parallel rays. Such 2D grids of rays tagged with intervals are known as a *ray-reps*.

In contrast to a regular 3D gridded volume, ray-reps have higher precision and generally lower memory footprint since they may be stored as a sequence of 2D depth images. In addition to storing intersection points on the boundary of the 3D shape, it is also possible to store auxiliary information with each intersection point, such as a normal or a material identifier. In this work, we derive a single, unified method that allows the accurate reconstruction of multi-material solids from a ray-rep tagged with normals and material identifiers.

Ray-reps are also closely related to two other representations: layered-depth images (LDI) in the graphics [72] and dexels in mechanical engineering [80, 39]. In graphics, LDIs have been applied in meshless renderings, transparency, CSG renderings, and collision detection [66, 20, 77, 36]. In mechanical engineering, ray-reps/dexels/LDIs

have been found to be useful for high-precision, interactive solid modeling [55, 39, 81] powerful due to the fact that CSG operations are simple to implement under ray-reps.

Reconstructing a solid representation from a ray-rep typically involves building a closed surface mesh that bounds each material encoded in the ray-rep. In most areas, generating a surface mesh from ray-reps consists of forming a quad from four neighboring intersection points in the ray-rep. However, constructing a closed mesh in the neighborhood of silhouettes on the ray-rep is not so straightforward. Most previous work on this conversion has focused mainly on three orthogonal ray-reps; known approaches form uniform grid interpretation of the ray-reps and apply grid-based contouring methods such as Marching Cubes and Dual Contouring onto the uniform grid [4, 49, 81].

In contrast, we focus on ray-reps from a single direction, and we propose a simple method that extracts the surface directly from the ray-rep samples without an intermediate grid volume representation. One direction ray-reps free our reconstruction method from the problem of inconsistency that can arise when three orthogonal ray-reps disagree at their common grid points due to numerical error. Generating one-direction ray-rep from a surface mesh is also faster than generating a three-direction ray-rep, making it more attractive to interactive applications or dynamically deforming meshes. Finally, we note that, in some applications such as seismic imaging, the ability to form three orthogonal ray-reps is impossible due to the limitations of the sensing technology. For example, a standard problem in seismic modeling to reconstruct a 3D approximation to the underlying multi-material geology using data collected from a set of bore holes.

5.1.1 Related Work

Ray representations have been studied in both graphics and engineering communities in different contexts. In graphics, the original LDI paper [72] considered generating

LDIs from multi-angled reference images to create simulated depth field for 2D photography. LDIs are extended to 3-directions (called layered depth cubes (LDC)) to create a view-independent scene for image-based rendering [49]. Pfister et al.'s work explored using LDC with additional information at each sample point for meshless rendering [66]. Everitt found that one-direction LDI can be used to create order-independent transparency [20]. One-direction LDI has also been used to perform collision detection [36]. More recently, Trapp et al. looked at using LDI to perform real-time volumetric tests for mesh models, which can be used to generate CSG and other types of renderings effects [77]. These previous works in graphics have not dealt with generating a surface mesh from LDI, which is the focus of our work.

In mechanical engineering, the community has first studied ray-reps in the more specific context of numerical-controlled (NC) milling [80, 39]. Van Hook first note that ray-rep (dexel) can be used in interactive milling display. Huang and Oliver then extended the ray-rep structure to allow multiple viewing angles in the display algorithm. In solid modeling, Menon et al. proposed ray-rep as a powerful alternative to boundary and CSG representations [55]. Benouamer and Michelucci proposed using a triple ray representation to allow accurate CSG modeling without using Breps or CSG data structures [4].

Our work is closely related to the work by Zhang et al [87]. Their work also focus on surface reconstruction from one-direction ray-reps. We will review their method for 2D, two-material reconstruction in the next section. However, we also provide extension of the 2D algorithm into 3D and multi-material domains, which are not available in their work. Furthermore, we will present a dual reconstruction that can be more flexible than their primal method. In another closely related work, Liu et al. present a reconstruction method for planar cross-sections [50]. We do not focus on this general case but instead study the problem of 2D grids of 1D intersections. This specialization enables us to have simple presentations and fast algorithms. We will further discuss both of these works and contrast with our works in the sections

to follow.

5.1.2 Contributions

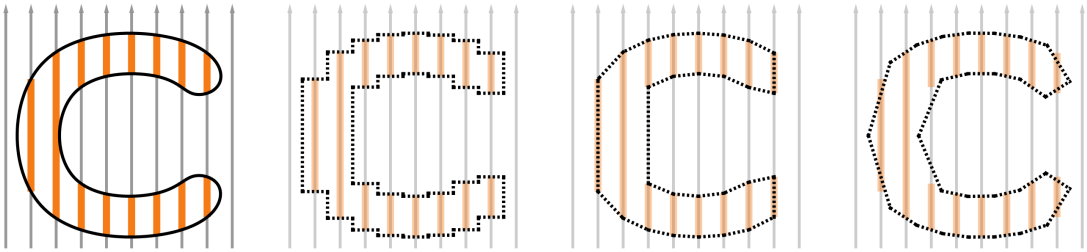
Our method builds on a 2D two-material reconstruction method of Zhang et al. [87]. As presented, this method uses a set of rules to generate edges connecting intersection points on adjacent rays in a ray-rep. Starting from this method,

- We re-formulate the method to yield a very simple 2D algorithm.
- We naturally extend the 2D algorithm to 3D.
- For ray-reps with normals, we describe a simple dual algorithm that uses normals to build a better approximation.
- For multiple material ray-reps, we extend the two-material dual algorithm to reconstruct multi-material geometry.

5.2 Reconstruction from two-material 2D ray-reps

We begin our investigation by considering the simplest case: a 2D ray-rep consisting of two materials. We will present a simple primal method (connecting intersection points on the rays) that is equivalent to Zhang et al.'s method and then present a dual method based on this primal method. We will also consider 3D versions of these two-material methods.

Given a closed 2D shape M, a ray-rep is usually generated by choosing a scan direction and intersecting a row of evenly-spaced *primary rays* parallel to the scan direction with M. For our purpose, we will view a ray-rep as an ordered sequence of pairs of intersection points attached to these ray. Figure 5.1(a) shows an example of a ray-rep for C-shaped area generated by choosing a vertical scan direction. Note that



(a) (b) (c) (d)

Figure 5.1 : A ray-rep computed from 2D shape (orange intervals) (a), its rectangular approximation(b), its primal reconstruction (c), and its dual reconstruction (d).

the problem of reconstructing this shape from the ray-rep is very similar to many 2D cross-section reconstruction problems.

5.2.1 Primal reconstruction

To aid in understanding our construction, we first form a rectangular approximation to M using an idea from Liu et al [50]. In particular, we will associate with each interval in the ray-rep a rectangle bounded horizontally by lines through its endpoints and vertically by lines midway between the rays containing the interval and its neighbors. Figure 5.1(b) shows such a rectangular approximation for the ray-rep of Figure 5.1(a). If we assign the material attached to each interval to its corresponding rectangle, the result is a partition of the area into a collection of rectangles whose union approximates M. Of course, this rectangular approximation has two obvious drawbacks: the edges in the approximation are all horizontal or vertical and adjacent rectangles of the same material share a common, redundant edge.

If each pair of adjacent primary rays bounds a *dual column* in the ray-rep, our goal is to generate a sequence of edges connecting the intersection points on this dual column that bounded M. Zhang et al. tackles this particular problem by independently considering each dual column and using a set of moderately complex rules to pair

intersection points. In the following, we will present an equivalent but simpler version of the Zhang et al.'s algorithm based on the rectangular approximation of Liu et al. We choose this particular presentation since it can be extend to both the dual case and the multi-material case in the later sections. Given a ray-rep whose rays are oriented vertically, the algorithm is as follows:

1. Horizontally project the intersection points on adjacent primal rays onto the centerline of the dual column and connect consecutive vertices on this centerline by vertical edges (Figure 5.2(a)),

2. Remove redundant vertical edges whose corresponding intervals on each primal ray are tagged with the same material (Figure 5.2(b)),

3. Construct edges connecting pairs of intersection points on the primary rays connected by vertical edges on the centerline (Figure 5.2(c)).

Note that step one constructs the restriction of Liu at al.'s rectangular approximation to the centerline of the dual column (Figure 5.2(a)). Step two of the method then deletes redundant vertical edges separating rectangles of the same material (Figure 5.2(b)). The output of step three are edges connecting intersection points of the original ray-rep (Figure 5.2(c)). These edges are the ones generated by Zhang et al.'s method. Figure 5.1(c) shows an example of this method applied to the ray-rep of Figure 5.1(a). Note that there are two types of edges produced by this method; *regular edges* that connect intersection points on adjacent primal rays and *silhouette edges* that connect intersection points lying on the same primal ray.

In the two-material case, the vertical edges generated by step one have an interesting property. *When vertically ordered, these edges strictly alternate between separating the two distinct materials and separating the same material.* This observation follows since each intersection point on a primary ray causes an alternation between the two materials. Thus, the effect of edge deletion of step two and the edge creation of step three can be summarized in the following elegant rule for directly generating edges:

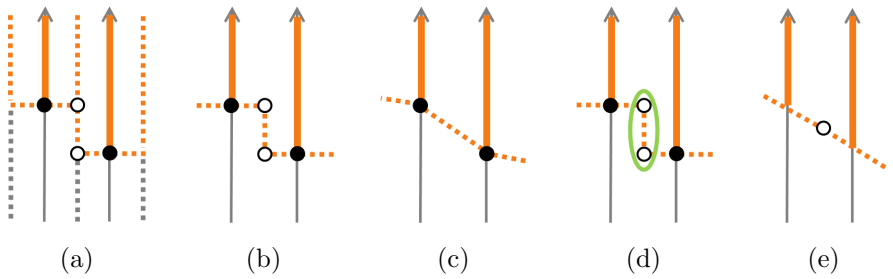


Figure 5.2 : Edge generation for a dual column. Creating a rectangular approximation (a), removing redundant vertical edges (b), connecting intersection point linked by a vertical edge (c), and merging a vertical edge to form a dual vertex (d) and (e).

> *Order the intersection points of each dual column vertically; generate edges connecting the first vertex to the second, the third to the fourth, and generally,* $2n - 1^{st}$ *to* $2n^{th}$.

Applying this method to each dual column of the ray-rep independently yields a set of edges forming a collection of closed curves. This observation follows since, for each dual column, step three generates exactly one edge incident on a given intersection point. Since each primary ray is shared by two dual columns, each intersection point is shared by exactly two edges generated in step three, resulting in a closed curve.

5.2.2 Dual reconstruction

One approach to improving the modeling capabilities of ray-reps is to augment intersection points by their associated surface normals [81]. We next describe a method for constructing dual meshes from ray-reps with normals that is similar in spirit to Dual Contouring [42]. This dual algorithm replaces each primal edge by a dual vertex positioned according to the normal data attached to endpoints of the primal edge and connects two dual vertices sharing a common primal vertex.

To aid our discussion of the multi-material case, we can formulate this dual method

directly in terms of the rectangular approximation used in our primal method. After step two of our primal construction (Figure 5.2(b)), we can topologically merge each vertical edge into a single dual vertex (Figures 5.2(d) and (e)). Observe that the effect of this merge is to remove all vertical edges from the rectangular approximation and create a final dual mesh formed entirely by deformed horizontal edges. Since each merged edge is the projection of two intersection points, we can the position of the dual resulting dual vertex to minimize the distance to the tangent lines as defined by the points and normals. (In 3D, the solution of this problem requires minimizing a quadratic functional and can be solved efficiently using standard linear techniques [71].) Figure 5.1(d) shows the dual mesh for the c-shape. Note that in comparison to the primal mesh, the dual mesh provides a better approximation near the silhouette of the shape.

5.3 Reconstruction from two-material 3D ray-reps

The structure of a three-dimensional ray-rep follows that of a two-dimensional ray-rep with the difference being that the one-dimensional sequence of primary rays is replaced by a two-dimensional rectangular grid of primary rays. In 3D, Zhang et al. uses contour tiling to reconstruct a 3D surface from a set of parallel 2D curves. In contrast, we present a simpler method to extract a 3D surface by applying our 2D method to pairs of adjacent rays in the ray-rep.

5.3.1 Primal reconstruction

As in 2D, our method process each dual column (the area bounded by four face-adjacent primary rays) independently. The method connects the intersection points on these four adjacent primary rays into a set of faces with the property that the union of these faces over all dual columns forms a closed mesh that bounds the underlying

shape. For each dual column, its associated faces will correspond to cycles formed by the edges lying on the two-dimensional faces of the column. After computing the edges on each face using our 2D method, we form the graph consisting of the union of these edges. To form faces, we then compute the edge cycles in this graph. This search consists of simple iterations through linear chains of edges that terminate each time a cycle is detected.

Figure 5.3 shows an example of four common cases processed by the method. The leftmost case shows two quad faces, each bounded by four regular edges. The middle left case shows a hexagonal face bounded by four regular edges and two silhouette edges. The middle right case shows another quad face bounded by two regular edges and two silhouette edges and the rightmost case show a degenerate two-sided face bounded by two silhouette edges. Note that other cases involving higher valence faces are possible.[1]

The key to the behavior of this face generation method is to observe that two dual columns sharing a common face also share the same edges on that face. This observation follows since these edges are determined solely by the intersection points on the two primary rays bounding the common face. Thus, every regular edge in the resulting mesh is shared by exactly two faces of the mesh and the resulting mesh is guaranteed to manifold across these edges. For silhouette edges, we observe that the primary ray containing this edge lies on four faces separating the four dual columns sharing that ray. On some faces, our 2D method generates this silhouette edges. On the others, the edge does not exists. For each of the faces containing this edge, our 3D method always generates two faces in the mesh (one in each dual column) that

[1]For example, consider the case in which the intervals associated with the four primary rays overlap in a pattern that spirals down the dual column. In this case, the edge generation method would generate two parallel sequences of regular edges that spiral around the dual column and that are connected by a silhouette edge near their top and bottom to form a single cycle. While arbitrarily large face valences are possible in theory, we have not observed this behavior in practice.

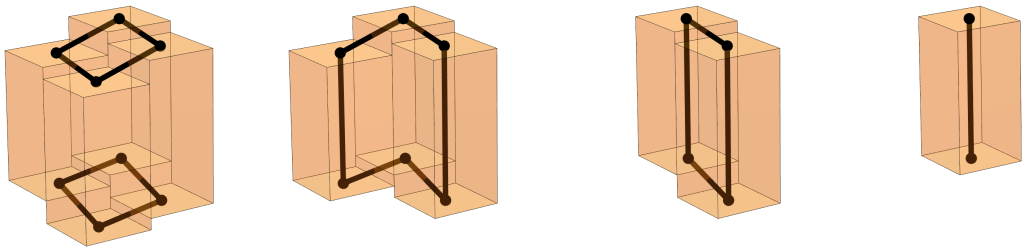


Figure 5.3 : Four common types of faces in 3D ray-rep meshes. Orange intervals lie on the four rays bounding a dual column. The blue edges form faces on the ray-rep mesh.

contain this edge. Thus, the edge valence of silhouette edges is always even and the overall mesh is guaranteed to be closed (but not necessarily manifold). Observe that this argument includes any degenerate 2-sided faces (as shown on the right of Figure 5.3) as being part of the topological mesh.

5.3.2 Dual reconstruction

We next consider a method for constructing two-material dual meshes in 3D. Given a 3D primal mesh generated by our method, note that every intersection point on the mesh lies on exactly four faces. Therefore, the corresponding dual mesh consists exclusively of quads. Our method for creating this dual meshes is as follow:

1. Construct edge cycles corresponding to faces in the primal 3D mesh,
2. For the points bounding each face, construct a single dual vertex that minimizes the distance from the tangent planes defined by these points and their normals,
3. Since each original intersection point on the ray-rep lies on four primal faces, generate a quad from its four associated dual vertices.

This simple construction requires applying the primal 3D mesh algorithm and identifying cycles of primal vertices in each dual column.

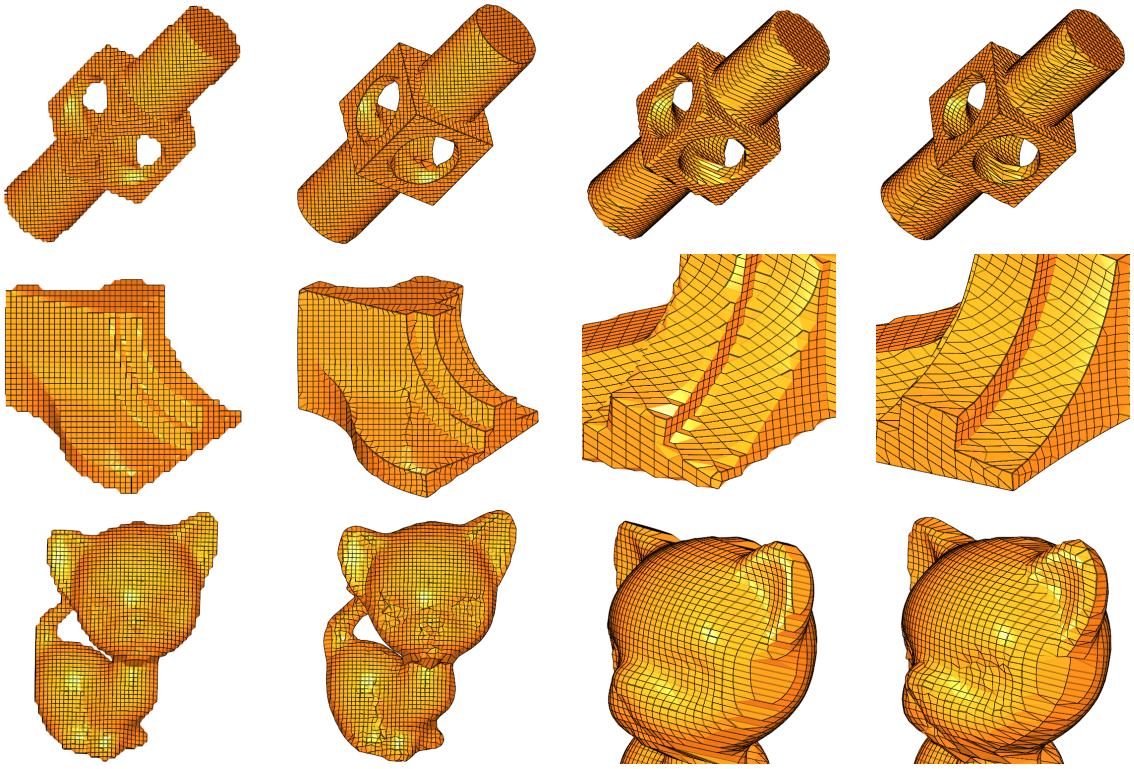


Figure 5.4 : Column (a) shows three examples of two-material primal reconstructions from ray-reps. (b) shows the dual reconstruction from the same models. (c) shows a side (silhouette) view of the meshes in (a). (d) shows the same side view for the dual meshes in (b). Note that, for ray-reps, sampling on the silhouette is typically poor, and the primal reconstruction reflects that lack of precision. In contrast, the dual meshes are able to reconstruct the sharp features.

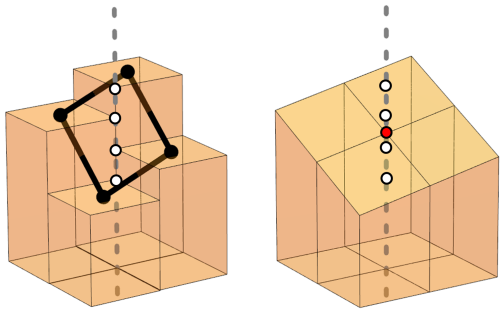
This dual mesh has an interesting interpretation in terms of the 3D generalization of Liu's rectangular approximation. For each interval on a vertical primary ray, we construct a rectangular prism bound above and below by horizontal square faces contain the endpoints of the interval and on its vertical sides by four rectangles whose vertical edges lie on the centerlines of the four dual columns containing the primary ray. Note the vertices of the prism all lie on the centerline of dual columns. In

particular, the prism vertices lying on the centerline for a dual column are just the horizontal project of the intersection points lying on the four primary rays bounding the dual column. As in 2D, the union of these rectangular prism form a prism approximation to the two materials encoded by the ray-rep. Figure 5.5(a) shows an example of this rectangular approximation for four rays forming a single dual column.

The topology of the dual mesh generated using the method described above has a simple interpretation in terms of this prism approximation. The quad faces in the final dual mesh correspond to the deformed horizontal faces in the 3D prism approximation. To form a closed mesh from these horizontal faces, sets of prism vertices lying on the centerline of dual column are merged by applying our 2D dual method to the faces of the dual column. In particular, each set of prism vertices corresponding to a primal face is merged to form a single dual vertex. For example, Figure 5.5(b) shows the four white prism vertices corresponding to a primal quad merged into a single dual vertex. Note that the vertical prism faces shared by different materials have their vertical edges collapsed to vertices by our 2D method causing the face to collapse to a line segment.

5.4 Reconstruction from multi-material ray-reps

We now consider the general case of the ray-rep reconstruction where intervals are tagged with three or more materials. In 2D, straightforward generalizations of our 2D primal and dual methods are available. In 3D, the primal method of tracing edge cycles to form faces leads to ambiguities in multi-material case. To avoid this problem, we generalize our 3D dual method for two-materials to the multi-material case.



(a) (b)

Figure 5.5 : The two-material dual construction using rectangular approximation. The black primal vertices are projected onto the centerline, denoted by white vertices (a). The four prism vertices on the centerline are used to create a single dual vertex (red vertex) (b).

5.4.1 Multi-material meshes in 2D

In 2D, the primal version of our multi-material algorithm is almost identical to the two-material case. As before, we focus on one dual column at a time and apply our three-step primal method. The first step forms the rectangular approximation of Liu et al (Figure 5.6(a)). The second step deletes vertical edges that separate rectangles of the same material (Figure 5.6(b)). The third step generates edges connecting intersection point on the primary rays whose corresponding projected (white) vertices share a vertical edge (Figure 5.6(c)).

Our 2D dual method uses the same two initial steps as our primal method, but instead topologically collapses sets of white vertices on the centerline of the dual column to form a dual mesh partitioning the materials (Figure 5.6(d)). In the two-material case, these remaining vertical edges had the elegant property that each projected vertex was incident on exactly one vertical edge allowing us to collapse these edge into a single valance dual vertex. In the multi-material case, a projected vertex can be shared by two vertical edges leading to connected chains of two or more consecutive

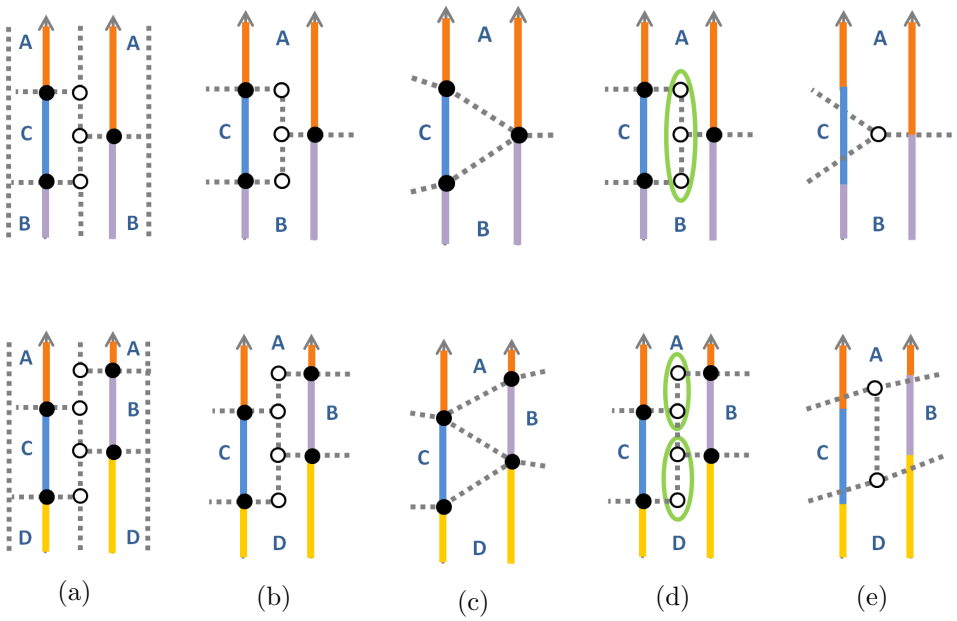
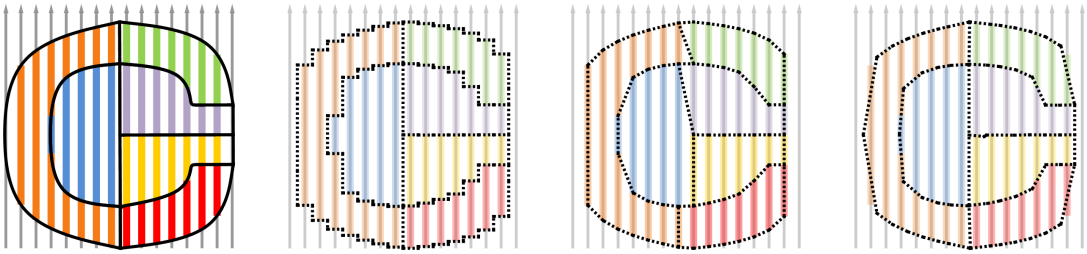


Figure 5.6 : Two examples of multi-material contouring algorithm. In the top row is an example with three materials, and in the bottom row is an example with four materials. (a) shows the rectangular approximation of the input ray-reps. (b) shows the removal of the redundant vertical edges. (c) shows the primal approximation of the input. (d) shows the merging of the prism vertices to generate a dual vertex in (e).

vertical edges. Our rule for merging these chains of edges is simple: topologically merge the vertices lying on the first and last edges of the chain. In particular, merge a chain of two vertical edges into a single valance three dual vertex (top of Figure 5.6(e)). For a chain of three vertical edges, we merge the first and last edge to form two valence three dual vertices connect a single dual edge (bottom of Figure 5.6(e)).

Note that this method preserves the topology of the rectangular approximation: if two materials border each other (either on an edge or vertex) in the rectangular approximation, then the two materials will border each other (at least on a vertex) in the final dual mesh. This property follows from the fact that merging first vertex



(a) (b) (c) (d)

Figure 5.7 : A 2D multi-material ray-rep (a), its rectangular approximation (b), its primal reconstruction (c), and its dual reconstruction (d). Notice that the primal reconstruction can zig-zag near the boundary between materials. In contrast, the dual reconstruction preserves the straight boundaries.

in a chain to the second vertex in the chain does not affect the topology of the solid. Figure 5.7 shows an example of 2D multi-material ray-rep as well as its rectangular approximation, its primal reconstruction and its dual reconstruction.

5.4.2 Multi-material meshes in 3D

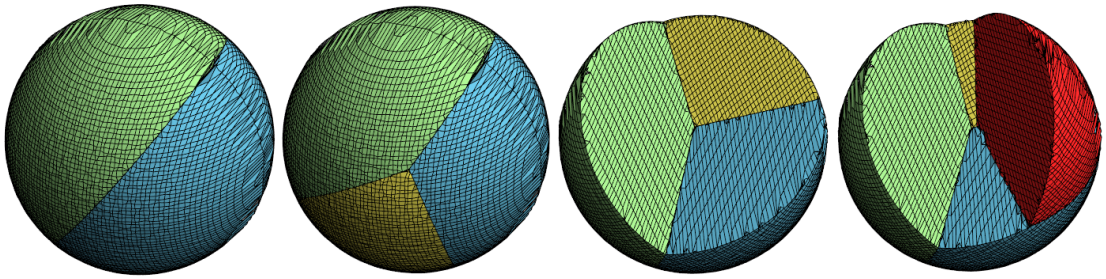
For two materials, the generalization of our primal method from 2D to 3D involves applying the 2D method to the faces of a single 3D dual column and tracing edge cycles to form faces associated with dual column. For three or more materials, this method produces a network of edges separating the various materials on the faces of the dual column. However, the problem of generating faces from this edge network is much more difficult due to the absence of easily identifiable cycles that form faces. Attempting to identify and consistently triangulate edge cycles in this multi-material edge network is quite challenging.

Instead, we focus our attention on constructing a 3D dual method for three or more materials that generalizes our two-material method. To this end, we begin with a solid approximation of a vertically-oriented ray-rep using rectangular prisms. The vertices

of this prism approximation lie on the centerlines of 3D dual columns. In particular, the vertices on the centerline of a single dual column are the horizontal projections of the intersection points on each of the primary rays bounding the column.

In the two-material case, we applied our dual method to each face of the dual column and merge pairs of vertices on the centerline on these faces. These merges operations were then applied to the projections of these vertices onto the centerline of the dual column. As seen in Figure 5.5, the four merges on the faces of the dual column lead to a single merged vertex on the centerline of the dual column. We take a similar approach in the multi-material case. We compute vertex merges on each face of the dual column using an extension of our 2D multi-material algorithm. We use a distance-based heuristic for merging vertices. Two vertices can be merged if they are joined by an edge, and the distance between the two vertices is below a certain threshold.

In the two-material case, the topological connectivity of these merged vertices was provided solely by the horizontal faces of the original prism approximation. All of the vertical faces in the prism approximation were collapsed into line segments due to vertex merges. In the multi-material case, these horizontal faces are also part of the final surface network used to partition the various materials. However, as opposed to the two-material case, this final surface network also includes some vertical faces inherited from the rectangular approximation. In particular, any vertical edges remaining after applying our extension of 2D dual method to a face of the 3D dual column corresponds to a vertical face separating two rectangular prism of different materials. These vertical faces appear in the final topology of the surface network (subject to vertex merges). The resulting merges on the faces of the dual column correspond to merging vertices on the vertical faces in the prism approximation. Each vertical face in the prism approximation is triangulated for rendering, and each pair of merged vertices in the prism approximation correspond to removing a triangle in the triangulation of the vertical face. Each final merged vertex on the centerline



(a) (b) (c) (d)

Figure 5.8 : 3D multi-material reconstructions with two materials, three materials, four materials (fourth material hidden), and five materials (fifth material hidden), respectively. All examples are sampled at a resolution of 80×80. Note that the silhouette suffers from under-sampling as is known for one-direction ray-reps.

is positioned using the position and normal data associated with its corresponding intersection points. Figure 5.8 shows few examples of 3D multi-material meshes.

5.5 Results and implementation

We implemented the two-material 3D meshing methods in C++ on an Intel Xeon 2.8GHz machine. Table 5.1 shows the timings for the various examples. (We compute 3D ray-reps from mesh models using depth peeling [20].) Interestingly, the dual reconstruction is generally faster than the primal reconstruction. There are two reasons for this: first, the primal method generate more triangles than the dual. Since each dual vertex corresponds to a primal face, which consist of at least four vertices, the number of dual triangles should be less than the primal. Second, the primal method requires special care in triangulation; note that we are generating edge cycles in the primal method. For cycles with more than three vertices, we need to determine a good triangulation of the vertices, which requires more computation. Hence, the dual method is typically faster than the primal.

model (resolution)	primal time(s)	primal tris	dual time(s)	dual tris
fandisk (50^2)	.025	5044	.024	4866
mechpart (100^2)	.093	18116	.0784	17322
mechpart (150^2)	.198	40836	.178	39590
kitten (200^2)	.337	64312	.277	63736
kitten (250^2)	.530	100416	.466	99716
kitten (300^2)	.753	144672	.605	143764

Table 5.1 : Timing results for ray-rep primal/dual mesh generation for the various models.

In multi-material case, we have implemented both our 2D and 3D dual methods in Mathematica. Given the similarity of our two-material and multi-material methods, we expect similar performance results for the multi-material case. We also note that our reconstruction methods are amenable to parallelization. Since all of the process takes place for each dual columns independently, we believe that our method can easily be parallelized using a language like CUDA.

5.6 Discussion

In this work, we have presented algorithms for building surfaces for two-material and multi-material ray-reps. We have shown that the 2D primal method of Zhang et al. has a very elegant description and can be easily extended to a primal 3D method. Furthermore, we use the rectangular approximation of ray-reps (described by Liu et al.) to examine the dual reconstruction methods for 2D and 3D. We also used the same rectangular approximation to generalize the two-material reconstructions to multiple materials. As demonstrated through examples, the dual mesh is more

flexible than the primal in cases where normal data is available.

For future work, we will examine the possibility extending our work for three direction ray-reps. Furthermore, we would like to study the relationship between ray-rep/LDIs and geometry images.

Chapter 6

Conclusion

In this thesis, we presented three representations of geometric shapes using functions. Each of the representations possesses properties that are desirable for particular computer graphics tasks. For example, B-splines are useful for representing smooth shapes in both 2D and 3D and for performing smooth image manipulation tasks. On the other hand, density maps and ray-reps are useful in representing complex shapes and in performing constructive solid geometric operations.

For B-spline, we studied the construction of the univariate B-spline, which has proven to be very useful in both the academia and the industry. In Chapter 2, our treatment of the univariate B-splines took a unique approach of looking at B-splines basis functions as solutions to differential equations. We showed that by taking integrals of the differential equations, we can obtain the standard B-spline properties of bump-likeness and locality. We also framed refinement of the B-spline as solutions to a linear system that correspond to differential conditions on the coarse and fine knot sets. Our univariate discussion also involved a brief description of building linear reproducing basis functions under this differential equation point of view.

In Chapter 3, we showed that the univariate construction can be extended to the bivariate case without much difficulty. Our univariate discourse enabled the construction of B-splines on two-manifolds, and we demonstrated this construction on the sphere. We believe that our approach of multivariate spline construction is only the beginning of research in this vein. In particular, we note that several important future directions of research are available, including building better discrete differ-

ential operators on the manifold, proving the convergence of the refinement scheme, and constructing polynomial reproducing basis functions in the bounded planar case.

In Chapter 4, we considered the case of visualizing a contour (level-set) in a density map. This visualization task is most often performed by extracting piece-wise linear primitives, such as triangles or quads, from the density map. Our work in this field concerned the case where the user wishes to remove invisible inner contour from the extraction process in order to reduce the rendering load. To that end, we proposed computing a Contour Visibility Function (CVF) by using a dynamic programming approach to approximate digital straight line visibility. Our algorithm guaranteed that the rendered results are the same for both the input and the CVF when viewed from a far-field position. Our algorithm is used as a preprocessing step for the density map, and therefore, does not incur any rendering time costs. Furthermore, by computing the CVF, our algorithm performs culling for all level-sets and for all angles in the view space.

Density maps continue to be an invaluable representation in scientific visualization and modeling. Our work in density map culling represents one in a myriad of manipulation techniques that are possible in the realm of density map representations. We note that density maps are often used in real-time rendering for games to approximate a global lighting effect called ambient occlusion. We propose that building stronger connections between ambient occlusion maps and constructing CVF as one possible direction of further research.

In Chapter 5, we examined the problem of extracting piece-wise linear primitives from ray-reps. Our goal in this work was to provide a general and elegant algorithm for contour extraction that parallels the development of Marching Cubes for density maps. In our work, we simplified previous results in 2D ray-rep contour extraction to build a local extraction scheme with very simple rules. Furthermore, we extended the 2D extraction into 3D and studied how hermite data at the intersection points

can be used to create better approximating contours. Lastly, we demonstrated how our algorithm enables for multiple-material extraction.

Ray-reps have been used for tasks in computer graphics such as transparency rendering and constructive solid geometry. Ray-rep finds its roots in mechanical engineering, where it remains a powerful representation for many mechanical modeling operations. In our work, we considered the specific case of one-direction ray-rep, but we note that most mechanical engineering works in this area have shifted towards multi-directional ray-reps. Whether our framework can be easily extended to multi-directional ray-reps remain an open question. In many ways, much of the elegance of our algorithm comes from the fact that one-direction ray-reps are conceptually simple and do not have the numerical inconsistency problem of multi-direction ray-reps. This is one of many possible future directions for ray-rep research in graphics. In addition, we believe that ray-rep is a very applicable tool in practice, and many of its potentials, including applications in destructible environments in games and animation, remains to be explored.

The field of geometric modeling has grown considerably in the past few decades due to advances in computing power. Creating complex 2D or 3D geometry is no longer a feat only possible for professionals with expensive resources. Instead, desktop computer, laptops, and even mobile devices have progressed such that shape creation tools are readily available to most people in the developed world. Our goal as computer graphics specialists is to provide efficient and powerful tools to enable people of various backgrounds to create and design as far as their imagination can take them. We hope the work described in this thesis contributed to the realization of such a goal.

Bibliography

[1] Andres, E., Acharya, R., and Sibata, C. Discrete analytical hyperplanes. *Graphical Models and Image Processing 59*, 5 (1997), 302 – 309.

[2] Bajaj, C. L., Coyle, E. J., and Lin, K.-N. Arbitrary topology shape reconstruction from planar cross sections. *Graph. Models Image Process. 58* (November 1996), 524–543.

[3] Beatson, R. K., and Light, W. A. Fast evaluation of radial basis functions: methods for two-dimensional polyharmonic splines. *IMA Journal of Numerical Analysis 17*, 3 (1997), 343–372.

[4] Benouamer, M. O., and Michelucci, D. Bridging the gap between csg and brep via a triple ray representation. In *Proceedings of the fourth ACM symposium on Solid modeling and applications* (New York, NY, USA, 1997), SMA '97, ACM, pp. 68–79.

[5] Boehm, W. Inserting new knots into b-spline curves. *Computer-Aided Design 12*, 4 (1980), 199–201.

[6] Boissonnat, J.-D. Shape reconstruction from planar cross sections. *Comput. Vision Graph. Image Process. 44* (August 1988), 1–29.

[7] Boissonnat, J.-D., and Cazals, F. Natural neighbor coordinates of points on a surface. *Computational Geometry 19*, 23 (2001), 155 – 173. Combinatorial Curves and Surfaces.

[8] Bozzola, J., and Russell, L. *Electron Microscopy: Pinciples and Techniques for Biologists*. Jones and Bartlett series in biology. Jones & Bartlett, 1999.

[9] Buhmann, M. D. *Radial basis functions: theory and implementations*, vol. 12. Cambridge Univ Pr, 2003.

[10] Buhmann, M. D., Dyn, N., and Levin, D. On quasi-interpolation by radial basis functions with scattered centres. *Constructive Approximation 11* (1995), 239–254.

[11] Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 67–76.

[12] Civril, A., Magdon-Ismail, M., and Bocek-Rivele, E. SDE: Graph drawing using spectral distance embedding. In *Graph Drawing* (2006), Springer, pp. 512–513.

[13] Cohen-Or, D., Chrysanthou, Y. L., Silva, C. T., and Durand, F. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics 9* (2003), 412–431.

[14] Dahmen, W., Micchelli, C. A., and Seidel, H. P. Blossoming begets B-spline bases built better by B-patches. *Mathematics of computation 59*, 199 (1992), 97–115.

[15] de Boor, C. *A Practical Guide to Splines*. No. v. 27 in Applied Mathematical Sciences. Springer, 2001.

[16] Desbrun, M., Kanso, E., and Tong, Y. Discrete differential forms for computational modeling. *Discrete differential geometry* (2008), 287–324.

[17] do Carmo, M. P. *Differential geometry of curves and surfaces*. Prentice-Hall Englewood Cliffs, NJ, 1976.

[18] Drebin, R. A., Carpenter, L., and Hanrahan, P. Volume rendering. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1988), SIGGRAPH '88, ACM, pp. 65–74.

[19] Dyn, N., Levin, D., and Rippa, S. Numerical procedures for surface fitting of scattered data by radial functions. *SIAM Journal on Scientific and Statistical Computing 7*, 2 (1986), 639–659.

[20] Everitt, C. Interactive Order-Independent Transparency, 2001.

[21] Farin, G. E. *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Code*, 4th ed. Academic Press, Inc., Orlando, FL, USA, 1996.

[22] Feng, P., Ju, T., and Warren, J. View-independent contour culling of 3d density maps for far-field viewing of iso-surfaces. *Computers & graphics 35*, 3 (2011), 561–568.

[23] Feng, P., and Warren, J. Discrete bi-laplacians and biharmonic b-splines. *ACM Trans. Graph. 31*, 4 (July 2012), 115:1–115:11.

[24] Feng, P., and Warren, J. A dual method for constructing multi-material solids from ray-reps. In *Advances in Visual Computing*. Springer, 2012, pp. 92–103.

[25] Fisher, M., Springborn, B., Schröder, P., and Bobenko, A. I. An algorithm for the construction of intrinsic Delaunay triangulations with applications to digital geometry processing. *Computing 81*, 2 (2007), 199–213.

[26] Fleming, W. H. *Functions of several variables*. Springer, 1977.

[27] Foley, J., Van Dam, A., Feiner, S., Hughes, J., and Phillips, R. *Introduction to computer graphics*, vol. 55. Addison-Wesley, 1994.

[28] Freeden, W., and Schreiner, M. *Spherical functions of mathematical geosciences: a scalar, vectorial, and tensorial setup*. Springer Verlag, 2009.

[29] Gao, J., Huang, J., Shen, H.-W., and Kohl, J. A. Visibility culling using plenoptic opacity functions for large volume visualization. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), IEEE Computer Society, p. 45.

[30] Gao, J., and Shen, H.-W. Parallel view-dependent isosurface extraction using multi-pass occlusion culling. In *PVG '01: Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics* (Piscataway, NJ, USA, 2001), IEEE Press, pp. 67–74.

[31] Gao, J., Shen, H.-W., Huang, J., and Kohl, J. A. Visibility culling for time-varying volume rendering using temporal occlusion coherence. In *VIS '04: Proceedings of the conference on Visualization '04* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 147–154.

[32] Garland, M., and Heckbert, P. S. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 209–216.

[33] Goldman, R. *Pyramid Algorithms: A Dynamic Programming Approach to Curves and Surfaces for Geometric Modeling.* The Morgan Kaufmann Series in Computer Graphics. Elsevier Science, 2002.

[34] Goldman, R., and Lyche, T. *Knot insertion and deletion algorithms for B-spline curves and surfaces.* No. 36. Society for Industrial Mathematics, 1993.

[35] Gregorski, B., Senecal, J., Duchaineau, M., and Joy, K. I. Compression and occlusion culling for fast isosurface extraction from massive datasets. *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration* (2009), 303–323.

[36] Heidelberger, B., Teschner, M., and Gross, M. Real-time volumet-

ric intersections of deforming objects. In *Proceedings of Vision, Modeling, and Visualization 2003* (Berlin, Germany, 2003), VMV '03, Akademische Verlagsgesellschaft Aka GmbH, Berlin, pp. 461–468.

[37] Herman, G. *Fundamentals of Computerized Tomography: Image Reconstruction from Projections*. Advances in pattern recognition. Springer London, 2009.

[38] Hiyoshi, H., and Sugihara, K. Voronoi-based interpolation with higher continuity. In *Proceedings of the sixteenth annual symposium on computational geometry* (New York, NY, USA, 2000), SCG '00, ACM, pp. 242–250.

[39] Huang, Y., and Oliver, J. H. Nc milling error assessment and tool path correction. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), SIGGRAPH '94, ACM, pp. 287–294.

[40] Jacobson, A., Tosun, E., Sorkine, O., and Zorin, D. Mixed finite elements for variational surface modeling. *Computer Graphics Forum (proceedings of EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing) 29*, 5 (2010), 1565–1574.

[41] John, F. *Partial Differential Equations*. No. v. 1 in Applied Mathematical Sciences. Springer-Verlag, 1982.

[42] Ju, T., Losasso, F., Schaefer, S., and Warren, J. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 339–346.

[43] Ju, T., Schaefer, S., and Warren, J. Mean value coordinates for closed triangular meshes. In *ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), SIGGRAPH '05, ACM, pp. 561–566.

[44] Klette, R., and Rosenfeld, A. Digital straightness–a review. *Discrete Ap-*

plied Mathematics 139, 1-3 (2004), 197 – 230. The 2001 International Workshop on Combinatorial Image Analysis.

[45] Lane, J. M., and Riesenfeld, R. F. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 1 (1980), 35–46.

[46] Lawson, C. L., Baker, M. L., Best, C., Bi, C., Dougherty, M., Feng, P., van Ginkel, G., Devkota, B., Lagerstedt, I., Ludtke, S. J., et al. Emdatabank.org: unified data resource for cryoem. *Nucleic acids research 39*, suppl 1 (2011), D456–D464.

[47] Li, W., Mueller, K., and Kaufman, A. Empty space skipping and occlusion clipping for texture-based volume rendering. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), IEEE Computer Society, p. 42.

[48] Lipman, Y., Rustamov, R. M., and Funkhouser, T. A. Biharmonic distance. *ACM Transactions on Graphics (TOG) 29*, 3 (2010), 27.

[49] Lischinski, B., and Rappoport, A. Image-based rendering for non-diffuse synthetic scenes. In *Rendering Techniques '98, Proceedings of the Eurographics Workshop* (1998), Springer, pp. 301–314.

[50] Liu, L., Bajaj, C., Deasy, J., Low, D., and Ju, T. Surface reconstruction from non-parallel curve networks. In *Computer Graphics Forum (Proceedings of Eurographics* (2008), vol. 27, John Wiley & Sons, pp. 155–163.

[51] Liu, Y., Chen, Z., and Tang, K. Construction of iso-contours, bisectors and Voronoi diagrams on triangulated surfaces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 99 (2011), 1–1.

[52] Livnat, Y., and Hansen, C. View dependent isosurface extraction. In *VIS '98: Proceedings of the conference on Visualization '98* (Los Alamitos, CA, USA,

1998), IEEE Computer Society Press, pp. 175–180.

[53] Loghin, D. *Green's functions for preconditioning.* PhD thesis, Oxford University, 1999.

[54] Lorensen, W. E., and Cline, H. E. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), SIGGRAPH '87, ACM, pp. 163–169.

[55] Menon, J., Marisa, R., and Zagajac, J. More powerful solid modeling through ray representations. *Computer Graphics and Applications, IEEE 14*, 3 (May 1994), 22 –35.

[56] Menon, J., and Voelcker, H. On the completeness and conversion of ray representations of arbitrary solids. In *Proceedings of the third ACM symposium on Solid modeling and applications* (1995), ACM, pp. 175–286.

[57] Monterde, J., and Ugail, H. On harmonic and biharmonic Bézier surfaces. *Computer Aided Geometric Design 21*, 7 (2004), 697–715.

[58] Mroz, L., Hauser, H., and Grller, E. Interactive high-quality maximum intensity projection. *Computer Graphics Forum 19*, 3 (2000), 341–350.

[59] Mroz, L., Konig, A., and Groller, E. Real-time maximum intensity projection. In *Data Visualization '99* (1999), Springer, pp. 135–144.

[60] Nay, R. A., and Utku, S. An alternative for the finite element method. *Variational Methods in Engineering 2* (Sept. 1972).

[61] Neamtu, M. Delaunay configurations and multivariate splines: A generalization of a result of BN Delaunay. *Transactions of the American Mathematical Society 359*, 7 (2007), 2993–3004.

[62] Neuman, E. Moments and fourier transforms of b-splines. *Journal of Compu-*

tational and Applied Mathematics 7, 1 (1981), 51 – 62.

[63] Nienhaus, M., and Dllner, J. *GPU Gems II: Programming Techniques for High Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, 2005, ch. Blueprint Rendering and Sketchy Drawings, pp. 235–252.

[64] Nirenstein, S., Blake, E., and Gain, J. Exact from-region visibility culling. In *Proceedings of the 13th Eurographics workshop on Rendering* (2002), EGRW '02, pp. 191–202.

[65] Pesco, S., Lindstrom, P., Pascucci, V., and Silva, C. T. Implicit occluders. *Volume Visualization and Graphics, IEEE Symposium on 0* (2004), 47–54.

[66] Pfister, H., Zwicker, M., van Baar, J., and Gross, M. Surfels: surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 335–342.

[67] Rabut, C. Elementary m-harmonic cardinal B-splines. *Numerical Algorithms 2*, 1 (1992), 39–61.

[68] Ramshaw, L. H. *Blossoming: A connect-the-dots approach to splines*. Digital Systems Research Center, 1987.

[69] Rose III, C. F., Sloan, P.-P. J., and Cohen, M. F. Artist-directed inverse-kinematics using radial basis function interpolation. *Computer Graphics Forum 20*, 3 (2001), 239–250.

[70] Roth, S. D. Ray casting for modeling solids. *Computer Graphics and Image Processing 18*, 2 (1982), 109 – 144.

[71] Schaefer, S., and Warren, J. Dual contouring: "the secret sauce". Rice University, Department of Computer Science Technical Report, 2003.

[72] Shade, J., Gortler, S., He, L.-w., and Szeliski, R. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 231–242.

[73] Sibson, R. A brief description of natural neighbour interpolation. *Interpreting multivariate data 21* (1981), 21–36.

[74] Slichter, C. *Principles of Magnetic Resonance*. Springer Series in Solid-State Sciences. Springer, 1996.

[75] Strikwerda, J. C. *Finite difference schemes and partial differential equations*. Society for Industrial Mathematics, 2004.

[76] Surazhsky, V., Surazhsky, T., Kirsanov, D., Gortler, S. J., and Hoppe, H. Fast exact and approximate geodesics on meshes. In *ACM Transactions on Graphics (TOG)* (2005), vol. 24, ACM, pp. 553–560.

[77] Trapp, M., and Dollner, J. Real-Time Volumetric Tests Using Layered Depth Images. *Eurographics 2008 Shortpaper* (2008), 235–238.

[78] Vallet, B., and Lévy, B. Spectral geometry processing with manifold harmonics. *Computer Graphics Forum 27*, 2 (2008), 251–260.

[79] Van De Ville, D., Blu, T., and Unser, M. Isotropic polyharmonic B-splines: scaling functions and wavelets. *Image Processing, IEEE Transactions on 14*, 11 (nov. 2005), 1798 –1813.

[80] Van Hook, T. Real-time shaded nc milling display. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1986), SIGGRAPH '86, ACM, pp. 15–20.

[81] Wang, C. C. L., Leung, Y.-S., and Chen, Y. Solid modeling of polyhedral objects by layered depth-normal images on the gpu. *Comput. Aided Des. 42* (June 2010), 535–544.

[82] Wardetzky, M., Mathur, S., Kälberer, F., and Grinspun, E. Discrete Laplace operators: no free lunch. In *Proceedings of the fifth Eurographics symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2007), SGP '07, Eurographics Association, pp. 33–37.

[83] Warren, J., and Weimer, H. *Subdivision methods for geometric design: a constructive approach*. Morgan Kaufmann series in computer graphics and geometric modeling. Morgan Kaufmann, 2002.

[84] Xu, G. Discrete Laplace-Beltrami operators and their convergence. *Computer Aided Geometric Design 21*, 8 (2004), 767–784.

[85] Xu, K., Zhang, H., Cohen-Or, D., and Xiong, Y. Dynamic harmonic fields for surface processing. *Computers & Graphics 33*, 3 (2009), 391–398.

[86] Zhang, H., Manocha, D., Hudson, T., and Hoff, III, K. E. Visibility culling using hierarchical occlusion maps. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 77–88.

[87] Zhang, W., Peng, X., Leu, M. C., and Zhang, W. A novel contour generation algorithm for surface reconstruction from dexel data. *Journal of Computing and Information Science in Engineering 7*, 3 (2007), 203–210.